LEVEL II ROMs

BY MARK D. GOODWIN

LEVEL II ROMs

No. 1575
$24.95

LEVEL II ROMs

BY MARK D. GOODWIN

TAB BOOKS Inc.
BLUE RIDGE SUMMIT, PA. 17214

To my princess and our wonderful family.

NOTICES

TRS-80 is a registered trademark of the Radio Shack Division of Tandy Corporation.
ESF and Exatron String Floppy are trademarks of the Exatron Corporation.
Epson MX-80 is a registered trademark of Epson America Inc.

FIRST EDITION

FIRST PRINTING

Printed in the United States of America

Library of Congress Cataloging in Publication Data

Gooodwin, Mark D. (Mark Dennis), 1956-
Level II Roms.

Includes index.
1. TRS-80 (Computer)—Programming. 2. Basic (Computer program language) 3. Read-only storage. I. Title.
II. Title: Level 2 ROMs. III. Title: Level two ROMs.
IV. Title: Level II R.O.M.s.
QA76.8.T18G66 1983 001.64'2 83-4931
ISBN 0-8306-0275-5
ISBN 0-8306-0175-9 (pbk.)

Contents

LIST OF PROGRAMS

*Type:

EDTASM—Requires the Radio Shack EDTASM. To load use the EDTASM's L command.

BASIC—To load use the CLOAD command.

Introduction

Ever since the TRS-80 Model I microcomputer was introduced to the computer world, there has been an astonishing interest in Z-80 assembly language programming. This great interest has led to much curiosity about the inner workings of the Level II BASIC ROMs. Programmers realized early on that there are many useful routines in the ROMs, and knowledge of these routines could greatly simplify the development of highly complex assembly language programs.

Although there have been many magazine articles and books written about the inner workings of the Level II BASIC ROMs, none of this information really presents the subject in a way that is useful to the Z-80 assembly language programmer. In writing this book I hope that I have solved this problem. Not only does this book present line-by-line comments for the Level II BASIC interpreter, it goes further by explaining how the powerful ROM routines can be put to practical use.

In addition to explaining the many useful ROM routines, the book presents many methods for interfacing machine language subroutines with Level II BASIC. It goes even further by explaining how commands can be added to the Level II BASIC interpreter and how existing Level II BASIC commands can be greatly enhanced. There is something in this book for anybody who has ever had an interest in learning more about the Level II BASIC ROMs. In closing, I hope you will have as much fun exploring the Level II BASIC ROMs as I have had writing about them.

Chapter 1
The Level II ROM Routines

The Z-80 microprocessor supports 8-bit and 16-bit addition and subtraction, but this is not a sufficient amount of precision for most program applications. Although it is possible to write routines to achieve higher precision, these routines are quite difficult to write and would require an extensive amount of program development time. So why not put the Level II math routines to work? After all, is there any sense in reinventing the wheel?

Level II BASIC supports three different types of numbers: integers, single precision numbers, and double precision numbers. This chapter deals with how these types of numbers can be put to use via the Level II ROM routines. The math routines as well as most of the important input/output routines are covered.

The highest amount of precision available on the Z-80 microprocessor is 16-bit. However, Level II BASIC's highest amount of precision, double precision numbers, uses 64 bits. As you can see, there is no way to manipulate the Level II high precision number types by using the Z-80s registers. Therefore, Level II BASIC must make certain areas of memory available as jumbo-sized registers. Level II BASIC uses two areas of memory for these registers. The locations are 411DH-4124H (hereinafter referred to as REG1) and 4127H-412EH (hereinafter referred to as REG2). Figure 1-1 shows how REG1 stores the three different types of Level II BASIC numbers. REG2 stores numbers exactly the same as REG1, but remember REG2 resides in a different area of memory.

Address	Integer	Single Precision	Double Precision
411DH			LSB
411EH			NMSB
411FH			NMSB
4120H			NMSB
4121H	LSB	LSB	NMSB
4122H	MSB	NMSB	NMSB
4123H		MSB	MSB
4124H		EXP	EXP

Fig. 1-1. REG1 memory locations. LSB = Least Significant Byte, NMSB = Next Most Significant Byte, MSB = Most Significant Byte.

In order for Level II BASIC to know which type of number is being used, it must use a number type flag (NTF). This number type flag is located in memory at 40AFH, and it must always hold the proper value necessary to reflect the current number type being used. The proper values for the number type flag are:

2	Integer
3	String
4	Single Precision
8	Double Precision

The rest of this chapter presents the major ROM routines one by one. Short examples are given where appropriate. A program at the end of the chapter makes use of some of these routines. You are encouraged to try all of the programs presented in this book; they will help you understand the topics discussed.

INTEGER ROUTINES

The integer math routines always use register pair HL and DE. The number type flag must be set for integer.

Integer Addition, address 0BD2H. This routine adds the value in register pair DE to the value in register pair HL. The result will be returned in register pair HL. Register pair DE will remain intact. If the result of the addition exceeds that of a legitimate integer value, the values will be converted to single precision, and the single precision result will be returned in REG1. For example:

```
LD      A,2         ;A=INTEGER NUMBER TYPE
LD      (40AFH),A   ;SET NTF TO INTEGER
LD      HL,60       ;LOAD HL WITH VALUE
LD      DE,200      ;LOAD DE WITH VALUE
CALL    0BD2H       ;ADD THEM HL=HL + DE
```

Integer Subtraction, address 0BC7H. This routine subtracts the value in register pair HL from the value in register pair DE. The result will be returned in register pair HL. Register pair DE will remain intact. If the result of the subtraction is less than that of a legitimate integer value, the values will be converted to single precision, and the single precision result will be returned in REG1. Example:

```
LD      A,2         ;A=INTEGER NUMBER TYPE
LD      (40AFH),A   ;SET NTF TO INTEGER
LD      DE,75       ;LOAD DE WITH VALUE
LD      HL,45       ;LOAD HL WITH VALUE
CALL    0BC7H       ;SUBTRACT THEM HL=DE - HL
```

Integer Multiplication, address 0BF2H. This routine multiplies the value in register pair DE by the value in register pair HL. The result is returned in register pair HL. Register pair DE will remain intact. If the result exceeds a legitimate integer value, the values will be converted to single precision, and the single precision result will be returned in REG1. Example:

```
LD      A,2          ;A=INTEGER NUMBER TYPE
LD      (40AFH),A    ;SET NTF TO INTEGER
LD      HL,30        ;LOAD HL WITH VALUE
LD      DE,-20       ;LOAD DE WITH VALUE
CALL    0BF2H        ;MULTIPLY THEM HL = DE * HL
```

Integer Division, address 2490H. This routine will divide the value in register pair DE by the value in register pair HL. Both values are first converted to single precision, and then the single precision values are divided. The result will be returned in REG1. For example:

```
LD      A,2          ;A=INTEGER NUMBER TYPE
LD      (40AFH),A    ;SET NTF TO INTEGER
LD      DE,1000      ;LOAD DE WITH VALUE
LD      HL,110       ;LOAD HL WITH VALUE
CALL    2490H        ;DIVIDE THEM REG1=DE / HL
```

Integer Compare, address 0A39H. This routine compares the value in register pair DE to the value in register pair HL. The result is returned in register A and the flags are set accordingly. The values returned in register A are:

If DE<HL then register A will be equal to 1
If DE=HL then register A will be equal to 0
If DE>HL then register A will be equal to −1

```
LD      A,2          ;A=INTEGER NUMBER TYPE
LD      (40AFH),A    ;SET NTF TO INTEGER
LD      HL,(FNUM)    ;LOAD HL WITH VALUE
LD      DE,(SNUM)    ;LOAD DE WITH VALUE
CALL    0A39H        ; COMPARE DE WITH HL
```

SINGLE PRECISION ROUTINES

All of the single precision math routines require one value to be in register pairs BC and DE. The other single precision value must

always be in REG1. The number type flag must be set for single precision.

Single Precision Addition, address 0716H. This routine will add the single precision value in register pairs BC and DE to the single precision value in REG1. The single precision result will be returned in REG1. Example:

```
LD      A,4          ;A=SINGLE PRECISION NUMBER TYPE
LD      (40AFH),A    ;SET NTF TO SINGLE PRECISION
LD      HL,NUM1      ;HL POINTS TO THE FIRST VALUE
CALL    09B1H        ;REG1=NUM1
LD      HL,NUM2      ;HL POINTS TO SECOND VALUE
CALL    09C2H        ;BCDE=NUM2
CALL    0716H        ;REG1=BCDE+REG1
```

Single Precision Subtraction, address 0713H. This routine subtracts the single precision value in REG1 from the single precision value in register pairs BC and DE. The single precision result will be returned in REG1. Example:

```
LD      A,4          ;A=SINGLE PRECISION NUMBER TYPE
LD      (40AFH),A    ;SET NTF TO SINGLE PRECISION
LD      HL,NUM1      ;HL POINTS TO FIRST VALUE
CALL    09B1H        ;REG1=NUM1
LD      HL,NUM2      ;HL POINTS TO SECOND VALUE
CALL    09C2H        ;BCDE=NUM2
CALL    0713H        ;REG1=BCDE-REG1
```

Single Precision Multiplication, address 0847H. This routine will multiply the single precision value in register pairs BC and DE by the single precision value in REG1. The single precision result will be returned in REG1. Example:

```
LD      A,4          ;A=SINGLE PRECISION NUMBER TYPE
LD      (40AFH),A    ;SET NTF TO SINGLE PRECISION
```

```
LD      HL,NUM1     ;HL POINTS TO FIRST VALUE
CALL    09B1H       ;REG1=NUM1
LD      HL,NUM2     ;HL POINTS TO SECOND VALUE
CALL    09C2H       ;BCDE=NUM2
CALL    0847H       ;REG1=BCDE*REG1
```

Single Precision Division, address 08A2H. This routine will divide the single precision value in register pairs BC and DE by the single precision value in REG1. The single precision result will be returned in REG1. For example:

```
LD      A,4         ;A=SINGLE PRECISION NUMBER TYPE
LD      (40AFH),A   ;SET NTF TO SINGLE PRECISION
LD      HL,NUM1     ;HL POINTS TO FIRST VALUE
CALL    09B1H       ;REG1=NUM1
LD      HL,NUM2     ;HL POINTS TO SECOND VALUE
CALL    09C2H       ;BCDE=NUM2
CALL    08A2H       ;REG1=BCDE/REG1
```

Single Precision Compare, address 0A0CH. This routine will compare the single precision value in register pairs BC and DE with the single precision value in REG1. The result will be returned in register A and the flags will be updated accordingly. The values returned in register A are:

If BCDE<REG1 then register A will be equal to 1
If BCDE=REG1 then register A will be equal to 0
If BCDE>REG1 then register A will be equal to −1

```
LD      A,4         ;A=SINGLE PRECISION NUMBER TYPE
LD      (40AFH),A   ;SET NTF TO SINGLE PRECISION
LD      HL,NUM1     ;HL POINTS TO FIRST VALUE
CALL    09B1H       ;REG1=NUM1
LD      HL,NUM2     ;HL POINTS TO SECOND VALUE
```

```
CALL    09C2H       ;BCDE=NUM2
CALL    0A0CH       ;COMPARE BCDE WITH REG1
```

DOUBLE PRECISION ROUTINES

All of the double precision math routines require one value to be in REG1. The other double precision value must always be in REG2. Furthermore, the number type flag must be set for double precision.

Double Precision Addition, address 0C77H. This routine will add the double precision value in REG1 to the double precision value in REG2. The double precision result will be returned in REG1. Example:

```
LD      A,8         ;A=DOUBLE PRECISION NUMBER TYPE
LD      (40AFH),A   ;SET NTF TO DOUBLE PRECISION
LD      DE,NUM1     ;DE POINTS TO FIRST VALUE
LD      HL,411DH    ;HL POINTS TO REG1
CALL    09D3H       ;REG1=FIRST VALUE
LD      DE,NUM2     ;DE POINTS TO SECOND VALUE
LD      HL,4127H    ;HL POINTS TO REG2
CALL    09D3H       ;REG2=SECOND VALUE
CALL    0C77H       ;REG1=REG1 REG2
```

Double Precision Subtraction, address 0C70H. This routine will subtract the double precision value in REG2 from the double precision value in REG1. The double precision result will be returned in REG1. For example:

```
LD      A,8         ;A=DOUBLE PRECISION NUMBER TYPE
LD      (40AFH),A   ;SET NTF TO DOUBLE PRECISION
LD      DE,NUM1     ;DE POINTS TO FIRST VALUE
LD      HL,411DH    ;HL POINTS TO REG1
CALL    09D3H       ;REG1=FIRST VALUE
LD      DE,NUM2     ;DE POINTS TO SECOND VALUE
```

```
LD      HL,4127H    ;HL POINTS TO REG2
CALL    09D3H       ;REG2=SECOND VALUE
CALL    0C70H       ;REG1=REG1+REG2
```

Double Precision Multiplication, address 0DA1H. This routine will multiply the double precision value in REG1 by the double precision value in REG2. The double precision result will be returned in REG1. Example:

```
LD      A,8         ;A=DOUBLE PRECISION NUMBER TYPE
LD      (40AFH),A   ;SET NTF TO DOUBLE PRECISION
LD      DE,NUM1     ;DE POINTS TO FIRST VALUE
LD      HL,411DH    ;HL POINTS TO REG1
CALL    09D3H       ;REG1=FIRST VALUE
LD      DE,NUM2     ;DE POINTS TO SECOND VALUE
LD      HL,4127H    ;HL POINTS TO REG2
CALL    09D3H       ;REG2=SECOND VALUE
CALL    0DA1H       ;REG1=REG1*REG2
```

Double Precision Division, address 0DE5H. This routine will divide the double precision value in REG1 by the double precision value in REG2. The double precision result will be returned in REG1. Example:

```
LD      A,8         ;A=DOUBLE PRECISION NUMBER TYPE
LD      (40AFH),A   ;SET NTF TO DOUBLE PRECISION
LD      DE,NUM1     ;DE POINTS TO FIRST VALUE
LD      HL,411DH    ;HL POINTS TO REG1
CALL    09D3H       ;REG1=FIRST VALUE
LD      DE,NUM2     ;DE POINTS TO SECOND VALUE
LD      HL,4127H    ;HL POINTS TO REG2
CALL    09D3H       ;REG2=SECOND VALUE
CALL    0DE5H       ;REG1=REG1/REG2
```

Double Precision Compare, address 0A4FH and 0A78H. These two routines will perform a double precision compare. The routine at 0A4FH will compare the double precision value in REG2 with the double precision value in REG1. The routine at 0A78H will compare the double precision value in REG1 with the double precision value in REG2. Both will return the result in register A and update the flags accordingly. The values returned in register A are:

0A4F		0A78	
REG2<REG1	A=1	REG1<REG2	A=1
REG2=REG1	A=0	REG1=REG2	A=0
REG2>REG1	A=−1	REG1>REG2	A=−1

```
LD      A,8         ;A=DOUBLE PRECISION NUMBER TYPE
LD      (40AFH),A   ;SET NTF TO DOUBLE PRECISION
LD      DE,NUM1     ;DE POINTS TO FIRST VALUE
LD      HL,411DH    ;HL POINTS TO REG1
CALL    09D3H       ;REG1=FIRST VALUE
LD      DE,NUM2     ;DE POINTS TO SECOND VALUE
LD      HL,4127H    ;HL POINTS TO REG2
CALL    09D3H       ;REG2=SECOND VALUE
CALL    0A4FH       ;COMPARE REG2 WITH REG1
```

MATH ROUTINES

All of the following math routines expect a value in REG1. The number type flag should properly reflect the number type in REG1.

Absolute Value ABS(REG1), address 0977H. This routine will figure the absolute value for the value in REG1. The result will be returned in REG1.

Arc tangent ATN(REG1), address 15BDH. This routine will figure the arc tangent for the value in REG1. The result will be returned in REG1.

Convert to Double Precision CDBL(REG1), address 0ADBH. This routine will convert the value in REG1 to a double precision value. The double precision result will be returned in REG1.

Convert to Integer CINT(REG1), address 0A7FH. This routine will convert the value in REG1 to an integer value. The integer result will be returned in REG1.

Cosine COS(REG1), address 1541H. This routine will figure the cosine for the value in REG1. The result will be returned in REG1.

Convert to Single Precision CSNG(REG1), address 0AB1H. This routine will convert the value in REG1 to a single precision value. The single precision result will be returned in REG1.

EXP(REG1), address 1439H. This routine will figure the natural exponential of the value in REG1. The result will be returned in REG1.

Stack↑REG1, address 13F2H. This routine will raise the single precision value on the stack to the single precision power in REG1. The single precision result will be returned in REG1.

FIX(REG1), address 0B26H. This routine will truncate the fractional part of the value in REG1. The result will be returned in REG1.

INT(REG1), address 0B37H. This routine will figure the integer portion of the value in REG1. The result will be returned in REG1.

Invert Sign: Integer, address 0C51H. This routine will invert the sign of an integer value in REG1. The integer result will be returned in REG1.

Invert Sign: Single Precision or Double Precision, address 0982H. This routine will invert the sign of a single precision or double precision value in REG1. The single precision or double precision result will be returned in REG1.

LOG(REG1), address 0809H. This routine will figure the natural log for the value in REG1. The result will be returned in REG1.

RANDOM, address 01D3H. This routine will reseed the random number generator.

RND(REG1), address 14C9H. This routine will generate a random number for the value in REG1. The result will be returned in REG1.

SIN(REG1), address 1547H. This routine will figure the sine for the value in REG1. The result will be returned in REG1.

SQR(REG1), address 13E7H. This routine will figure the square root for the value in REG1. The result will be returned in REG1.

TAN(REG1), address 15A8H. This routine will figure the tangent for the value in REG1. The result will be returned in REG1.

KEYBOARD INPUT ROUTINES

The following routines will make inputting data from the keyboard much easier.

Scan Keyboard, address 002BH. This routine will scan the keyboard. If a key isn't pressed, register A will return with a value of zero and the flags will be updated accordingly. If a key is pressed, register A will return with the ASCII value for the key that was pressed. Example:

```
;WAIT TILL KEY PRESSED
          PUSH      DE        ;SAVE DE
          PUSH      IY        ;SAVE IY
LOOP      CALL      002BH     ;SCAN KEYBOARD
          JR        Z,LOOP    ;IF NO KEY PRESSED LOOP
          POP       IY        ;GET IY
          POP       DE        ;GET DE
```

Wait Till Key Pressed, address 0049H. This routine will continually scan the keyboard until a key is pressed. The ASCII value for the appropriate key will be returned in register A.

Get a Line of Input from the Keyboard, address 0361H. This routine will accept keyboard input and place each character in a buffer. Input is terminated by pressing either the ENTER key or the BREAK key. On return, register pair HL will contain the address of the first character input minus one. If the BREAK key is pressed, the Carry flag will be set.

Get a Line of Input from the Keyboard, address 1BB3H. This routine functions the same as the routine at 0361H except that a ? will be displayed before input begins.

OUTPUT ROUTINES

The following routines allow for output to the video display, to the printer, and to the cassette. The current output device is determined by the value stored at memory location 409CH. The correct values are:

1	Printer
0	Video
−1	Cassette

The current output device will normally be set for the video display.

Display the Character in Register A, address 0033H. This routine will display the character in register A at the current cursor location.

Clear the Screen, address 01C9H. This routine will clear the screen and home the cursor.

Display the Character in Register A, address 032AH. This routine will display the character in register A at the current cursor location. Furthermore, the current cursor location will be updated.

Display or Print a String, address 28A7H. This routine will send a string of text to the current output device. On entry to this routine, register pair HL should point to the start of the string to be sent. The string should be terminated with either a zero (00H) or a carriage return (0DH). The current output device flag at 409CH must be set to 0 for video or 1 for printer.

Display or Print a String, address 2B75H. This routine is the same as the routine at 28A7H except that the string can only be terminated by a zero (00H). The current output device flag at 409CH must be set to 0 for video or 1 for printer.

Print the Character in Register C, address 003BH. This routine will print the character in register C on the printer.

CASSETTE INPUT/OUTPUT ROUTINES

The following routines allow for input and output on the cassette recorder. The current output device flag at 409CH must be set to −1.

Turn on the Cassette Motor, address 0212H. This

routine will turn on the motor for the cassette drive specified by the value in register A.

Write Cassette Leader, address 0284H. This routine will write a leader on the cassette recorder.

Read Cassette Leader, address 0296H. This routine will read a leader on the cassette recorder.

Read a Byte, address 0235H. This routine will read a byte on the cassette recorder and place it in register A.

Write a Byte, address 0264H. This routine will write the byte in register A on the cassette recorder.

DATA MOVEMENT ROUTINES

The following routines are used to move data around in memory.

Move REG1 to Stack, address 09A4H. This routine will move the value in REG1 to the stack. The number of bytes moved is determined by the current value of the number type flag. This routine can only be used for integer and single precision values.

Move (HL) to REG1, address 09B1H. This routine will move the single precision value pointed to by register pair HL to REG1. The number type flag should be set for single precision.

Move BCDE to REG1, address 09B4H. This routine will move the single precision value in register pairs BC and DE to REG1. The number type flag should be set for single precision.

Move REG1 to BCDE, address 09BFH. This routine will move the single precision value in REG1 to register pairs BC and DE. The number type flag should be set for single precision.

Move (HL) to BCDE, address 09C2H. This routine will move the single precision value pointed to by register pair HL to register pairs BC and DE. The number type flag should be set for single precision.

Move REG1 to (HL), address 09CBH. This routine will move the single precision value in REG1 to the location pointed to by register pair HL. The number type flag should be set for single precision.

Move (DE) to (HL), address 09CEH. This routine will move the single precision value pointed to by register

pair DE to the location pointed to by register pair HL. The number type flag should be set for single precision.

Move (HL) to (DE), address 09D2H. This routine will move the value pointed to by register pair HL to the location pointed to by register pair DE. The number of bytes moved is determined by the value of the number type flag.

Move (DE) to (HL), address 09D3H. This routine will move the value pointed to by register pair DE to the location pointed to by register pair HL. The number of bytes moved is determined by the value of the number type flag.

Move (DE) to (HL), address 09D6H. This routine will move the area of memory pointed to by register pair DE to the location pointed by register pair H. The number of bytes moved is determined by the value in register A.

Move (DE) to (HL), address 09D7H. This routine will move the area of memory pointed to by register pair DE to the location pointed to by register pair HL. The number of bytes moved is determined by the value in register B.

Move REG2 to REG1, address 09F4H. This routine moves the value in REG2 to REG1.

Move REG1 to REG2, address 09FCH. This routine moves the value in REG1 to REG2.

ASCII CONVERSION ROUTINES

The following routines allow for ASCII conversions.

ASCII to Binary, address 0E6CH. This routine will convert the ASCII string pointed to by register pair HL to binary. The result will be returned in REG1, and the number type flag will be updated accordingly. The routine will convert the ASCII string to the least amount of precision required.

ASCII to Double Precision, address 0E65H. This routine will convert the ASCII string pointed to by register pair HL to a double precision value. The double precision result will be returned in REG1, and the number type flag will be updated accordingly.

ASCII to Integer, address 1E5AH. This routine will convert the ASCII string pointed to by register pair HL to

an integer value. The integer result will be returned in register pair DE.

Binary to ASCII, address 0FBDH. This routine will convert the value in REG1 to an ASCII string. On return, register pair HL will point to the first character of the ASCII string.

RESTART ROUTINES

The following routines are all RSTs and can be quite useful when trying to interface a routine with Level II BASIC.

Check Syntax, RST 0008H. This routine will compare the byte pointed to by register pair HL with the byte following the RST 0008H call. If the two bytes match, register pair HL will be incremented and register A will contain the byte pointed to by register pair HL. Execution will resume at the location following the byte which was compared.

Get Next Character, RST 0010H. This routine will increment register pair HL and place the byte pointed to by register pair HL in register A. If the character in register A is alphabetic, the Carry flag will be cleared. If the character in register A is numeric, the Carry flag will be set. If the character in register A is a control code or a space, the routine will start over.

Compare DE to HL, RST 0018H. This routine will perform an unsigned compare of register pairs DE and HL. On return, the flags will be set as follows:

HL<DE	Carry Set
HL>DE	Carry Clear
HL=DE	Zero Set
HL<>DE	Zero Clear

A SAMPLE PROGRAM

While there are many more useful routines in Level II BASIC, the routines presented in this chapter are the ones that will be most often put to use in a variety of program applications. More specialized routines are presented in future chapters when necessary.

The program presented in Listing 1-1 is a simple four function calculator. It will perform addition, subtraction, multiplication, and

division. Furthermore, it will perform all of these operations in integer, single precision, and double precision number representations. The program makes use of many of the ROM routines presented in this chapter and will help you understand these powerful routines.

Listing 1-1

```
4300                        ORG     4300H
4300 CDC901    START        CALL    01C9H
4303 216144                 LD      HL,M1
4306 CD752B                 CALL    2B75H
4309 218E44    LOOP1        LD      HL,M2
430C CD752B                 CALL    2B75H
430F CD4900                 CALL    0049H
4312 214744                 LD      HL,CTAB
4315 0603                   LD      B,3
4317 BE        LOOP2        CP      (HL)
4318 CA2243                 JP      Z,MATCH
431B 23                     INC     HL
431C 23                     INC     HL
431D 23                     INC     HL
431E 10F7                   DJNZ    LOOP2
4320 18E7                   JR      LOOP1
4322 23        MATCH        INC     HL
4323 5E                     LD      E,(HL)
4324 23                     INC     HL
4325 56                     LD      D,(HL)
4326 EB                     EX      DE,HL
4327 E9                     JP      (HL)
4328 CD2D44    INT          CALL    INPUT
432B CD7F0A                 CALL    0A7FH
432E 2A2141                 LD      HL,(4121H)
4331 225144                 LD      (NUM1),HL
4334 CD2D44                 CALL    INPUT
4337 CD7F0A                 CALL    0A7FH
433A 2A2141                 LD      HL,(4121H)
433D 225944                 LD      (NUM2),HL
4340 CD3A44                 CALL    FUN
4343 2A5144                 LD      HL,(NUM1)
4346 EB                     EX      DE,HL
4347 2A5944                 LD      HL,(NUM2)
434A 3A5044                 LD      A,(FUNSAV)
434D FE2B                   CP      '+'
434F 280E                   JR      Z,INTA
4351 FE2D                   CP      '-'
4353 280F                   JR      Z,INTS
4355 FE2A                   CP      '*'
4357 2810                   JR      Z,INTM
4359 FE2F                   CP      '/'
435B 2811                   JR      Z,INTD
435D 18AA                   JR      LOOP1
435F CDD20B    INTA         CALL    0BD2H
4362 180D                   JR      DISP
4364 CDC70B    INTS         CALL    0BC7H
4367 1808                   JR      DISP
4369 CDF20B    INTM         CALL    0BF2H
436C 1803                   JR      DISP
436E CD9024    INTD         CALL    2490H
4371 CDBD0F    DISP         CALL    0FBDH
4374 CD752B                 CALL    2B75H
4377 3E0D                   LD      A,0DH
4379 CD2A03                 CALL    032AH
```

```
437C 188B                 JR      LOOP1
437E CD2D44         SIN   CALL    INPUT
4381 CDB10A               CALL    0AB1H
4384 215144               LD      HL,NUM1
4387 CDCB09               CALL    09CBH
438A CD2D44               CALL    INPUT
438D CDB10A               CALL    0AB1H
4390 215944               LD      HL,NUM2
4393 CDCB09               CALL    09CBH
4396 CD3A44               CALL    FUN
4399 215944               LD      HL,NUM2
439C CDB109               CALL    09B1H
439F 215144               LD      HL,NUM1
43A2 CDC209               CALL    09C2H
43A5 3A5044               LD      A,(FUNSAV)
43A8 FE2B                 CP      '+'
43AA 280F                 JR      Z,SINA
43AC FE2D                 CP      '-'
43AE 2810                 JR      Z,SINS
43B0 FE2A                 CP      '*'
43B2 2811                 JR      Z,SINM
43B4 FE2F                 CP      '/'
43B6 2812                 JR      Z,SIND
43B8 C30943               JP      LOOP1
43BB CD1607         SINA  CALL    0716H
43BE 18B1                 JR      DISP
43C0 CD1307         SINS  CALL    0713H
43C3 18AC                 JR      DISP
43C5 CD4708         SINM  CALL    0847H
43C8 18A7                 JR      DISP
43CA CDA208         SIND  CALL    08A2H
43CD 18A2                 JR      DISP
43CF CD2D44         DOU   CALL    INPUT
43D2 CDDB0A               CALL    0ADBH
43D5 211D41               LD      HL,411DH
43D8 115144               LD      DE,NUM1
43DB CDD209               CALL    09D2H
43DE CD2D44               CALL    INPUT
43E1 CDDB0A               CALL    0ADBH
43E4 211D41               LD      HL,411DH
43E7 115944               LD      DE,NUM2
43EA CDD209               CALL    09D2H
43ED CD3A44               CALL    FUN
43F0 215144               LD      HL,NUM1
43F3 111D41               LD      DE,411DH
43F6 CDD209               CALL    09D2H
43F9 215944               LD      HL,NUM2
43FC 112741               LD      DE,4127H
43FF CDD209               CALL    09D2H
4402 3A5044               LD      A,(FUNSAV)
4405 FE2B                 CP      '+'
4407 280F                 JR      Z,DOUA
4409 FE2D                 CP      '-'
440B 2811                 JR      Z,DOUS
440D FE2A                 CP      '*'
440F 2812                 JR      Z,DOUM
4411 FE2F                 CP      '/'
4413 2813                 JR      Z,DOUD
4415 C30943               JP      LOOP1
4418 CD770C         DOUA  CALL    0C77H
441B C37143         DIS1  JP      DISP
441E CD700C         DOUS  CALL    0C70H
4421 18F8                 JR      DIS1
4423 CDA10D         DOUM  CALL    0DA1H
4426 18F3                 JR      DIS1
4428 CDE50D         DOUD  CALL    0DE5H
442B 18EE                 JR      DIS1
442D 21D244         INPUT LD      HL,M3
```

```
4430 CD752B          CALL    2B75H
4433 CDB31B          CALL    1BB3H
4436 D7              RST     0010H
4437 C36C0E          JP      0E6CH
443A 21DE44    FUN   LD      HL,M4
443D CD752B          CALL    2B75H
4440 CD4900          CALL    0049H
4443 325044          LD      (FUNSAV),A
4446 C9              RET
4447 49        CTAB  DEFB    'I'
4448 2843            DEFW    INT
444A 53              DEFB    'S'
444B 7E43            DEFW    SIN
444D 44              DEFB    'D'
444E CF43            DEFW    DOU
4450 00        FUNSAV DEFB   0
0008           NUM1  DEFS    8
0008           NUM2  DEFS    8
4461 46        M1    DEFM    'FOUR FUNCTION CALCULATOR
4462 4F
4463 55
4464 52
4465 20
4466 46
4467 55
4468 4E
4469 43
446A 54
446B 49
446C 4F
446D 4E
446E 20
446F 43
4470 41
4471 4C
4472 43
4473 55
4474 4C
4475 41
4476 54
4477 4F
4478 52
4479 0D              DEFB    13
447A 42              DEFM    'BY MARK D. GOODWIN'
447B 59
447C 20
447D 4D
447E 41
447F 52
4480 4B
4481 20
4482 44
4483 2E
4484 20
4485 47
4486 4F
4487 4F
4488 44
4489 57
448A 49
448B 4E
448C 0D              DEFB    13
448D 00              DEFB    0
448E 50        M2    DEFM    'PRESS:'
448F 52
4490 45
4491 53
```

```
4492 53
4493 3A
4494 0D          01570          DEFB      13
4495 49         01580          DEFM      'I FOR INTEGER'
4496 20
4497 46
4498 4F
4499 52
449A 20
449B 49
449C 4E
449D 54
449E 45
449F 47
44A0 45
44A1 52
44A2 0D         01590          DEFB      13
44A3 53         01600          DEFM      'S FOR SINGLE PRECISION'
44A4 20
44A5 46
44A6 4F
44A7 52
44A8 20
44A9 53
44AA 49
44AB 4E
44AC 47
44AD 4C
44AE 45
44AF 20
44B0 50
44B1 52
44B2 45
44B3 43
44B4 49
44B5 53
44B6 49
44B7 4F
44B8 4E
44B9 0D         01610          DEFB      13
44BA 44         01620          DEFM      'D FOR DOUBLE PRECISION'
44BB 20
44BC 46
44BD 4F
44BE 52
44BF 20
44C0 44
44C1 4F
44C2 55
44C3 42
44C4 4C
44C5 45
44C6 20
44C7 50
44C8 52
44C9 45
44CA 43
44CB 49
44CC 53
44CD 49
44CE 4F
44CF 4E
44D0 0D         01630          DEFB      13
44D1 00         01640          DEFB      0
44D2 45         01650 M3       DEFM      'ENTER VALUE'
44D3 4E
44D4 54
44D5 45
```

```
44D6 52
44D7 20
44D8 56
44D9 41
44DA 4C
44DB 55
44DC 45
44DD 00        01660          DEFB      0
44DE 50        01670 M4       DEFM      'PRESS:'
44DF 52
44E0 45
44E1 53
44E2 53
44E3 3A
44E4 0D        01680          DEFB      13
44E5 2B        01690          DEFM      '+ FOR ADDITION'
44E6 20
44E7 46
44E8 4F
44E9 52
44EA 20
44EB 41
44EC 44
44ED 44
44EE 49
44EF 54
44F0 49
44F1 4F
44F2 4E
44F3 0D        01700          DEFB      13
44F4 2D        01710          DEFM      '- FOR SUBTRACTION'
44F5 20
44F6 46
44F7 4F
44F8 52
44F9 20
44FA 53
44FB 55
44FC 42
44FD 54
44FE 52
44FF 41
4500 43
4501 54
4502 49
4503 4F
4504 4E
4505 0D        01720          DEFB      13
4506 2A        01730          DEFM      '* FOR MULTIPLICATION'
4507 20
4508 46
4509 4F
450A 52
450B 20
450C 4D
450D 55
450E 4C
450F 54
4510 49
4511 50
4512 4C
4513 49
4514 43
4515 41
4516 54
4517 49
4518 4F
4519 4E
```

```
451A 0D        01740          DEFB     13
451B 2F        01750          DEFM     '/ FOR DIVISION'
451C 20
451D 46
451E 4F
451F 52
4520 20
4521 44
4522 49
4523 56
4524 49
4525 53
4526 49
4527 4F
4528 4E
4529 0D        01760          DEFB     13
452A 00        01770          DEFB     0
4300           01780          END      START
00000 TOTAL ERRORS

M4       44DE
M3       44D2
DIS1     441B
DOUD     4428
DOUM     4423
DOUS     441E
DOUA     4418
DOU      43CF
SIND     43CA
SINM     43C5
SINS     43C0
SINA     43BB
SIN      437E
DISP     4371
INTD     436E
INTM     4369
INTS     4364
INTA     435F
FUNSAV   4450
FUN      443A
NUM2     4459
NUM1     4451
INPUT    442D
INT      4328
MATCH    4322
LOOP2    4317
CTAB     4447
M2       448E
LOOP1    4309
M1       4461
START    4300
```

Listing 1-1 Comments

100 - Set the starting address for the program.
110 - Go clear the screen.
120 - Load register pair HL with the starting address for the message to be displayed.
130 - Go display the message.
140 - Load register pair HL with the starting address for the message to be displayed.
150 - Go display the message.

160 - Go get a character from the keyboard and return with it in register A.

170 - Load register pair HL with the starting address for the command jump table.

180 - Load register B with the number of commands in the command jump table.

190 - Check to see if the character at the location of the command table jump pointer in register pair HL is the same as the character in register A.

200 - Jump if the character at the location of the command jump table pointer in register pair HL is the same as the character in register A.

210 - Bump the command jump table pointer in register pair HL.

220 - Bump the command jump table pointer in register pair HL.

230 - Bump the command jump table pointer in register pair HL.

240 - Loop till all of the commands in the command jump table have been examined.

250 - Jump if the character in register A doesn't match any of the commands in the command jump table.

260 - Bump the command jump table pointer in register pair HL.

270 - Load register E with the LSB of the command's starting address at the location of the command jump table pointer in register pair HL.

280 - Bump the command jump table pointer in register pair HL.

290 - Load register D with the MSB of the command's starting address at the location of the command jump table pointer in register pair HL.

300 - Load register pair HL with the command's starting address in register pair DE.

310 - Jump to the command's starting address in register pair HL.

320 - Go get the first value.

330 - Convert the first value in REG1 to an integer.

340 - Load register pair HL with the integer value in REG1.

350 - Save the integer value in register pair HL.

360 - Go get the second value.

370 - Convert the second value in REG1 to an integer.

380 - Load register pair HL with the integer value in REG1.

390 - Save the integer value in register pair HL.

400 - Go get the function to be performed.

410 - Load register pair HL with the first integer value.

420 - Load register pair DE with the first integer value in register pair HL.

430 - Load register pair HL with the second integer value.
440 - Load register A with the function to be performed.
450 - Check to see if the function to be performed in register A is addition.
460 - Jump if the function to be performed in register A is addition.
470 - Check to see if the function to be performed in register A is subtraction.
480 - Jump if the function to be performed in register A is subtraction.
490 - Check to see if the function to be performed in register A is multiplication.
500 - Jump if the function to be performed in register A is multiplication.
510 - Check to see if the function to be performed in register A is division.
520 - Jump if the function to be performed in register A is division.
530 - Jump if the function to be performed in register A isn't any of the four allowable functions.
540 - Go add the integer value in register pair HL to the integer value in register pair DE and return with the result in REG1.
550 - Jump.
560 - Go subtract the integer value in register pair HL from the integer value in register pair DE and return with the result in REG1.
570 - Jump.
580 - Go multiply the integer value in register pair DE by the integer value in register pair HL and return with the result in REG1.
590 - Jump.
600 - Go divide the integer value in register pair DE by the integer value in register pair HL and return with the result in REG1.
610 - Go convert the result in REG1 to an ASCII string.
620 - Go display the ASCII string.
630 - Load register A with a carriage return character.
640 - Go display the carriage return.
650 - Jump.
660 - Go get the first value.
670 - Convert the first value in REG1 to a single precision value.

680 - Load register pair HL with the storage address for the single precision value in REG1.
690 - Go move the single precision value in REG1.
700 - Go get the second value.
710 - Convert the second value in REG1 to a single precision value.
720 - Load register pair HL with the storage address for the single precision value in REG1.
730 - Go move the single precision value in REG1.
740 - Go get the function to be performed.
750 - Load register pair HL with the storage address for the second single precision value.
760 - Go move the second single precision value into REG1.
770 - Load register pair HL with the storage address for the first single precision value.
780 - Go move the first single precision value into register pairs BC and DE.
790 - Load register A with the function to be performed.
800 - Check to see if the function to be performed in register A is addition.
810 - Jump if the function to be performed in register A is addition.
820 - Check to see if the function to be performed in register A is subtraction.
830 - Jump if the function to be performed in register A is subtraction.
840 - Check to see if the function to be performed in register A is multiplication.
850 - Jump if the function to be performed in register A is multiplication.
860 - Check to see if the function to be performed in register A is division.
870 - Jump if the function to be performed in register A is division.
880 - Jump if the function to be performed in register A isn't any of the four allowable functions.
890 - Go add the single precision value in REG1 to the single precision value in register pairs BC and DE. Return with the result in REG1.
900 - Jump.
910 - Go subtract the single precision value in REG1 from the

single precision value in register pairs BC and DE. Return with the result in REG1.

920 - Jump.

930 - Go multiply the single precision value in register pairs BC and DE by the single precision value in REG1. Return with the result in REG1.

940 - Jump.

950 - Go divide the single precision value in register pairs BC and DE by the single precision value in REG1.

960 - Jump.

970 - Go get the first value.

980 - Convert the first value in REG1 to a double precision value.

990 - Load register pair HL with the starting address of REG1.

1000 - Load register pair DE with the storage address for the double precision value in REG1.

1010 - Go move the double precision value in REG1.

1020 - Go get the second value.

1030 - Convert the second value in REG1 to a double precision value.

1040 - Load register pair HL with the starting address of REG1.

1050 - Load register pair DE with the storage address for the double precision value in REG1.

1060 - Go move the double precision value in REG1.

1070 - Go get the function to be performed.

1080 - Load register pair HL with the storage address for the first double precision value.

1090 - Load register pair DE with the starting address of REG1.

1100 - Go move the first double precision value into REG1.

1110 - Load register pair HL with the storage address for the second double precision value.

1120 - Load register pair DE with the starting address of REG2.

1130 - Go move the second double precision value into REG2.

1140 - Load register A with the function to be performed.

1150 - Check to see if the function to be performed in register A is addition.

1160 - Jump if the function to be performed in register A is addition.

1170 - Check to see if the function to be performed in register A is subtraction.

1180 - Jump if the function to be performed in register A is subtraction.

1190 - Check to see if the function to be performed in register A is multiplication.
1200 - Jump if the function to be performed in register A is multiplication.
1210 - Check to see if the function to be performed in register A is division.
1220 - Jump if the function to be performed in register A is division.
1230 - Jump if the function to be performed in register A isn't any of the four allowable functions.
1240 - Go add the double precision value in REG2 to the double precision value in REG1 and return with the result in REG1.
1250 - Jump.
1260 - Go subtract the double precision value in REG2 from the double precision value in REG1 and return with the result in REG1.
1270 - Jump.
1280 - Go multiply the double precision value in REG1 by the double precision value in REG2 and return with the result in REG1.
1290 - Jump.
1300 - Go divide the double precision value in REG1 by the double precision value in REG2 and return with the result in REG1.
1310 - Jump.
1320 - Load register pair HL with the starting address of the message to be displayed.
1330 - Go display the message.
1340 - Go get the input from the keyboard.
1350 - Bump the input buffer pointer in register pair HL till it points to the first character.
1360 - Go convert the ASCII string at the location of the input buffer pointer in register pair HL to binary and return with the result in REG1.
1370 - Load register pair HL with the starting address of the message to be displayed.
1380 - Go display the message.
1390 - Go wait till a key is pressed and return with the ASCII value for the key in register A.
1400 - Save the function to be performed in register A.
1410 - Return.

1420 - The integer command is stored here.
1430 - The integer command jump address is stored here.
1440 - The single precision command is stored here.
1450 - The single precision command jump address is stored here.
1460 - The double precision command is here.
1470 - The double precision command jump address is stored here.
1480 - The function to be performed is stored here.
1490 - The first value will be stored here.
1500 - The second value input will be stored here.
1510 - Part of the sign-on message is stored here.
1520 - This will display a carriage return as part of the sign-on message.
1530 - Part of the sign-on message is stored here.
1540 - This will display a carriage return as part of the sign-on message.
1550 - The sign-on message terminator is stored here.
1560 - Part of the command message is stored here.
1570 - This will display a carriage return as part of the command message.
1580 - Part of the command message is stored here.
1590 - This will display a carriage return as part of the command message.
1600 - Part of the command message is stored here.
1610 - This will display a carriage return as part of the command message.
1620 - Part of the command message is stored here.
1630 - This will display a carriage return as part of the command message.
1640 - The command message terminator is stored here.
1650 - The value prompting message is stored here.
1660 - The value prompting message terminator is stored here.
1670 - Part of the function message is stored here.
1680 - This will display a carriage return as part of the function message.
1690 - Part of the function message is stored here.
1700 - This will display a carriage return as part of the function message.
1710 - Part of the function message is stored here.
1720 - This will display a carriage return as part of the function message.
1730 - Part of the function message is stored here.

1740 - This will display a carriage return as part of the function message.
1750 - Part of the function message is stored here.
1760 - This will display a carriage return as part of the function message.
1770 - The function message terminator is stored here.
1780 - End of the program.

Chapter 2
Machine Language Subroutines

Perhaps the easiest method of interfacing a machine language subroutine with Level II BASIC is by using the USR command. Although there are many more advanced techniques for linking machine language with Level II BASIC, they are incomprehensible until you understand the USR command, an essential first step.

Let's take a second to review the scanty information contained in the Level II Reference Manual about the USR command. With Level II BASIC there is only one USR call allowed. Before calling the subroutine, the starting address must be poked into memory. Suppose that there is a machine language subroutine at 27648. To tell BASIC where the starting address is, POKE16526,0 and POKE16527,108 would have to be executed. If these pokes were a part of a BASIC program, they might appear as:

```
10 POKE16526,0:POKE16527,108
```

To call this routine, the program would have to contain a line similar to

```
20 X=USR(0)
```

If a value is to be passed to the machine language subroutine, lines such as the following might be used.

```
20 X=USR(100)
100 Z=USR(I(5))
```

To pass the value to the machine language subroutine, a call to 0A7FH must be executed by the subroutine. This routine places the value into register pair HL.

To return from a machine language subroutine, simply put a RET instruction in the subroutine at the point where BASIC execution is to be resumed. To pass a value back to BASIC and return, load register pair HL with the value to be passed back and execute a JP to 0A9AH. To illustrate how values are passed back and forth, let's take a look at the following:

```
100 A=USR(25):PRINTA
```

The value of 25 is to be passed to the machine language subroutine. The subroutine is

```
CALL        0A7FH       ;HL=VALUE PASSED
ADD         HL,HL       ;HL=HL*2
JP          0A9AH       ;RETURN TO BASIC WITH VALUE
```

A blow-by-blow description of this example, would go something like this. As soon as BASIC encounters the USR command in line number 100, program control is given to the machine language subroutine. The call to 0A7FH will place the value passed, 25, in register pair HL. Next by adding register pair HL with itself, register pair HL will contain 50. Now the program executes a JP to 0A9AH, which in turn, makes the variable A equal to 50. Furthermore, the JP to 0A9AH resumes BASIC execution. This in turn will cause a 50 to appear on the video display.

Now that the use of USR has been covered, it's time to take a look at how subroutines can be loaded into memory. For the rest of the chapter, we will take a look at the various methods for loading subroutines. These methods include loading the subroutine from cassette, poking the subroutine into high memory, packing the subroutine into an array, and packing the subroutine into a string.

A GRAPHICS SUBROUTINE

Before we look at subroutine loading methods, we need a subroutine. The subroutine used to demonstrate the various methods of subroutine loading is one that will reverse all of the graphic characters on the video display.

In order to write this subroutine, you must understand the

Bit No.	7	6	5	4	3	2	1	0	
	1	0	1	1	1	1	1	1	= 191

Graphic Character

0	1
2	3
4	5

The video display is broken down into 64 spaces by 16 lines. Each space can be further broken down into a 2 by 3 graphic character. Thus the effective graphics resolution on the Model I is 128 by 48 graphic points. Each of these points is called a pixel. Each pixel corresponds with a particular bit of video memory. Shown above is what a byte would look like if 191 were poked into video memory. Also shown is an enlarged graphic character which has each pixel numbered. These numbers correspond directly with the bits in video memory. Any bit with a value of 1 will be on and any bit with a value of 0 will be off. Therefore, if 191 were poked into video memory; a graphic character with all of it's pixels on would be displayed. So the value for any graphic character can be determined as follows:

Bit 0 or Pixel 0	To set add a value of 1
Bit 1 or Pixel 1	To set add a value of 2
Bit 2 or Pixel 2	To set add a value of 4
Bit 3 or Pixel 3	To set add a value of 8
Bit 4 or Pixel 4	To set add a value of 16
Bit 5 or Pixel 5	To set add a value of 32
Bit 6	Always off
Bit 7	Always on, add a value of 128

Thus, if we wished to display a graphic character with pixels 0, 3 and 4 on, we would determine the code to be used like this.

Pixel 0 -	1
Pixel 3 -	8
Pixel 4 -	16
Bit 7 -	128
	153

Fig. 2-1. Model I graphic codes.

Model I's video display system. The Model I uses a memory-mapped video display system. The video memory starts at address 3C00H (15360) and is 1024 bytes long. Each byte of video memory holds one character. The value of each byte is determined by the ASCII code for the character to be displayed. If you were to POKE15360,90, a Z would be displayed in the upper left-hand corner of the video display. The codes for graphic characters are any value from 128 to 191. Figure 2-1 illustrates how the code is determined for a particular graphic character.

Now that graphic codes have been illustrated, it is an easy matter to determine how the graphic display can be reversed. We know that if a pixel is on, the corresponding bit is equal to 1; and when a pixel is off, the corresponding bit is equal to 0. Therefore, we must change all 0's to 1's and all 1's to 0's in bits 0 through 5 of each graphic character in video memory. Also, bit 6 must be off and bit 7 must be on for each graphic character. Figure 2-2 illustrates the three machine language instructions necessary to complete the reversal process.

Now that the method for graphics reversal is clear, it is an easy matter to write the subroutine. Listing 2-1 presents a program that reverses all graphic characters on the video display. The program will leave all non-graphic characters intact.

Listing 2-1

```
7FE6              00100          ORG     7FE6H
7FE6  21FF3B      00110  REVER   LD      HL,3BFFH
7FE9  23          00120          INC     HL
7FEA  01FF03      00130          LD      BC,03FFH
7FED  03          00140          INC     BC
7FEE  7E          00150  LOOP    LD      A,(HL)
7FEF  FE80        00160          CP      80H
7FF1  3806        00170          JR      C,NGRAPH
7FF3  2F          00180          CPL
7FF4  CBFF        00190          SET     7,A
7FF6  CBB7        00200          RES     6,A
7FF8  77          00210          LD      (HL),A
7FF9  23          00220  NGRAPH  INC     HL
7FFA  0B          00230          DEC     BC
7FFB  78          00240          LD      A,B
7FFC  B1          00250          OR      C
7FFD  20EF        00260          JR      NZ,LOOP
7FFF  C9          00270          RET
7FE6              00280          END     REVER
00000 TOTAL ERRORS

NGRAPH   7FF9
LOOP     7FEE
REVER    7FE6
```

```
;REGISTER A CONTAINS THE VALUE 191 or 10111111
        CPL                 ;A NOW HOLDS 01000000
        SET     7,A         ;A NOW HOLDS 11000000
        RES     6,A         ;A NOW HOLDS 10000000
```

Fig. 2-2. Graphics reversal.

Listing 2-1 Comments

100 - Set the starting address for the program.
110 - Load register pair HL with the start of video memory minus one.
120 - Bump the video memory pointer in register pair HL.
130 - Load register pair BC with the length of video memory minus one.
140 - Bump the video memory counter in register pair BC.
150 - Load register A with the character at the location of the video memory pointer in register pair HL.
160 - Check to see if the character in register A is a graphic character.
170 - Jump if the character in register A isn't a graphic character.
180 - Compliment the character in register A.
190 - Set bit 7 of the character in register A.
200 - Reset bit 6 of the character in register A.
210 - Display the graphic character in register A at the location of the video memory pointer in register pair HL.
220 - Bump the video memory pointer in register pair HL.
230 - Decrement the video memory counter in register pair BC.
240 - Load register A with the MSB of the video memory counter in register B.
250 - Combine the LSB of the video memory counter in register C with the MSB of the video memory counter in register A.
260 - Jump if this isn't the end of the video memory.
270 - Return.
280 - End of the program.

A close inspection of this program reveals something a little strange at the start of the program. As stated earlier, video memory starts at 3C00H (15360) and is 0400H (1024) bytes in length. So why does the program load register pair HL with 3BFFH and register pair BC with 03FFH? Normally lines 110 and 130 would be written

```
110         REVER       LD          HL,3C00H
130                     LD          BC,0400H
```

Lines 120 and 140 would also be deleted. However, if the program were to be assembled with these changes, the object code for these lines would be

```
110  - 21003C
130  - 010004
```

Note that both of these machine language instructions will result in two bytes of zeros (00H) in the object code. While this is acceptable for most applications, it would cause problems if this subroutine were packed into a string. These problems will be discussed later in this chapter. For now it is only necessary to understand that by loading register pair HL with 3BFFH and then incrementing it, register pair HL will contain 3C00H. Furthermore, by loading register pair BC with 03FFH and incrementing it, register pair BC will contain 0400H. Thus, the zero bytes are avoided in the object code, and the register pairs HL and BC contain the correct values after incrementing.

THE CASSETTE METHOD

The simplest method for interfacing this subroutine with BASIC is to load the subroutine into high memory via the cassette. To do this, a copy of the object code for the program in Listing 2-1 must be made on cassette. Once the object code has been produced, memory size must be set at 32742 (7FE6H), and the subroutine must be loaded with the SYSTEM command.

Next a BASIC program is needed, in order to make use of the machine language subroutine. Listing 2-2 contains a short BASIC program that first draws alternating horizontal lines on the video display. Once the display has been completed, the reversal subroutine is called over and over again giving the effect of the lines bouncing up and down.

Listing 2-2

```
0 ' LISTING 2.2
5 ' SUBROUTINE SHOULD BE LOADED AND MEM SIZE = 32742
10 ' DISPLAY ALTERNATING LINES
15 CLS
20 FORY=0TO47STEP2
```

```
30 FORX=0TO127
40 SET(X,Y)
50 NEXTX
60 NEXTY
70 ' WAIT FOR SHORT DELAY
80 FORI=1TO100
90 NEXTI
100 ' SET STARTING ADDRESS
110 POKE16526,230:POKE16527,127
120 ' REVERSE THE GRAPHICS
130 I=USR(0)
140 ' WAIT FOR SHORT DELAY
150 FORI=1TO100
160 NEXTI
170 'KEEP DOING UNTIL BREAK PRESSED
180 GOTO80
```

THE POKE METHOD

Although the cassette method for interfacing a machine language subroutine is quite easy, it can be quite a pain. Remember, each time the BASIC program is to be used, memory size? must be set and the subroutine must be loaded from cassette.

The poke method for loading the subroutine eliminates the need for a cassette recorder. In order to use the poke method, the object code must be converted to it's decimal equivalent. This conversion is necessary because the subroutine will be stored in the program data statements. Figure 2-3 shows the object code's decimal equivalent.

Now that we know the decimal equivalent for the subroutine's object code, add the following lines to the program appearing in Listing 2-2:

```
6 FORM=32742TO32767

7 READB

8 POKEM,B

9 NEXTM

1000 DATA33,255,59,35,1,255,3,3,126,254,128,56,6,47

1010 DATA203,255,203,183,119,35,11,120,177,32,239,201
```

Once these changes are completed, the program should look like Listing 2-3. What these new lines do is really quite simple. Lines 6 to 9 poke the values read from the data statements into high memory. So by using the poke method, the need for loading the machine language subroutine from cassette is completely eliminated. However, the need for setting memory size is still there.

Hex	Decimal	Hex	Decimal
21	33	2F	47
FF	255	CB	203
3B	59	FF	255
23	35	CB	203
01	1	B7	183
FF	255	77	119
03	3	23	35
03	3	0B	11
7E	126	78	120
FE	254	B1	177
80	128	20	32
38	56	EF	239
06	6	C9	201

Fig. 2-3. Decimal equivalents for the object code in Listing 2-1.

Listing 2-3

```
0 ' LISTING 2.3
5 ' POKE SUBROUTINE INTO HIGH MEMORY
6 FORM=32742TO32767
7 READB
8 POKEM,B
9 NEXTM
10 ' DISPLAY ALTERNATING LINES
15 CLS
20 FORY=0TO47STEP2
30 FORX=0TO127
40 SET(X,Y)
50 NEXTX
60 NEXTY
70 ' WAIT FOR SHORT DELAY
80 FORI=1TO100
90 NEXTI
100 ' SET STARTING ADDRESS
110 POKE16526,230:POKE16527,127
120 ' REVERSE THE GRAPHICS
130 I=USR(0)
140 ' WAIT FOR SHORT DELAY
150 FORI=1TO100
160 NEXTI
170 ' KEEP DOING UNTIL BREAK PRESSED
180 GOTO80
1000 DATA33,255,59,35,1,255,3,3,126,254,128,56,6,47
1010 DATA203,255,203,183,119,35,11,120,177,32,239,201
```

THE ARRAY PACKING METHOD

The necessity for setting memory size can be eliminated by interfacing the machine language subroutine with a method called array packing.

In order to change the program in Listing 2-3 to an array packing program, add

```
1 DEFINTA-Z
2 DIMS(12)
```

These lines initialize all variables as integers and array S with 13 elements. The reason for setting array S at 13 elements is that the machine language subroutine is 26 bytes long. Since an integer variable is 2 bytes long you divide 26 by 2 and get 13. Thus, the array to be packed must have at least 13 elements.

Array Element	Memory Location	Contents
S(0) LSB	17152	0
S(0) MSB	17153	0
S(1) LSB	17154	0
S(1) MSB	17155	0
S(2) LSB	17156	0
S(2) MSB	17157	0
S(3) LSB	17158	0
S(3) MSB	17159	0
S(4) LSB	17160	0
S(4) MSB	17161	0
S(5) LSB	17162	0
S(5) MSB	17163	0
S(6) LSB	17164	0
S(6) MSB	17165	0
S(7) LSB	17166	0
S(7) MSB	17167	0
S(8) LSB	17168	0
S(8) MSB	17169	0
S(9) LSB	17170	0
S(9) MSB	17171	0
S(10) LSB	17172	0
S(10) MSB	17173	0
S(11) LSB	17174	0
S(11) MSB	17175	0
S(12) LSB	17176	0
S(12) MSB	17177	0

Fig. 2-4. An integer array containing 13 elements.

Figure 2-4 will help you understand why packing the array is possible. Note that all elements in the array take up a consecutive area in memory. Although an integer array will be used in our program, a single precision array or a double precision array could be used. To figure the number of elements necessary for a single precision array, you simply divide the number of bytes in the subroutine by four. Remember, a single precision variable requires 4 bytes of memory. The size of the array is figured by dividing 26 by 4, which equals 6.5. Because an array can't have half an element, the array must be dimensioned for 7 elements. This example demonstrates an important point. Although a single precision array can be used, the extra array element wastes memory space. Therefore, an integer array should be used to conserve memory space.

To pack the array, the program will have to be changed as follows:

```
6 FORM=0TO25

8 POKEVARPTR(S(0))+M,B

110 I=VARPTR(S(0)):POKE16526,I-INT(I/256)*256:POKE
    16527,INT(I/256)
```

Once these changes are made, the program will look like the one in Listing 2-4. The VARPTR(S(0)), in lines 8 and 110 locates the starting address for the subroutine. After the packing has been completed, array S will appear in memory as shown in Fig. 2-5.

Once the array has been packed, the program must never use array S as a normal variable. Just consider what would happen if the program contained a line like 11 S(5)=15.

This line changes the contents of element 5 in array S, and the machine language program would not function properly if the subroutine were called.

Listing 2-4

```
0 ' LISTING 2.4
1 DEFINTA-Z
2 DIMS(12)
5 ' POKE SUBROUTINE INTO ARRAY S()
6 FORM=0TO25
7 READB
8 POKEVARPTR(S(0))+M,B
9 NEXTM
10 ' DISPLAY ALTERNATING LINES
15 CLS
20 FORY=0TO47STEP2
30 FORX=0TO127
```

```
40 SET(X,Y)
50 NEXTX
60 NEXTY
70 ' WAIT FOR SHORT DELAY
80 FORI=1TO100
90 NEXTI
100 ' SET STARTING ADDRESS
110 I=VARPTR(S(0)):POKE16526,I-INT(I/256)*256:POKE16527,
     INT(I/256)
120 ' REVERSE THE GRAPHICS
130 I=USR(0)
140 ' WAIT FOR SHORT DELAY
150 FORI=1TO100
160 NEXTI
170 ' KEEP DOING UNTIL BREAK PRESSED
180 GOTO80
1000 DATA33,255,59,35,1,255,3,3,126,254,128,56,6,47
1010 DATA203,255,203,183,119,35,11,120,177,32,239,201
```

THE STRING PACKING METHOD

The last method of interfacing machine language with BASIC is called string packing. Although string packing takes a little more program development time, it has many advantages over the other methods. It eliminates the need for cassette loading and the necessity of setting memory size. It also speeds up program execution time and is more memory efficient than the three previous methods because the string packing routine and the data statements can be eliminated after packing is completed. String packing does have one disadvantage though. You can't pack 00H into the string because zero is used by Level II BASIC as the BASIC line terminator. So, if you want to use string packing in your programs, avoid zeros. Beat around the bush, as in the graphics reversal program.

To change the array packing program to a string packing program, delete line 2 and change the following lines:

```
1 S$="..........................":

8 I=VARPTR(S$):SA=PEEK(I+2)*256+PEEK(I+1):POKESA+M,B

9 NEXTM:STOP

110 I=VARPTR(S$):POKE16526,PEEK(I+1):POKE16527,
    PEEK(I+2)
```

Note that in line 1, S$ is set to a string constant of 26 periods. The reason for this is that the machine language subroutine to be packed is 26 bytes long. It doesn't really matter if you use 26 periods or 26 other characters, just so long as S$ is 26 bytes in length.

In order to understand why string packing is a very desirable method for interfacing machine language with BASIC, it is first necessary to understand how a string is stored in memory. All

Array Element	Memory Location	Contents
S(0) LSB	17152	33
S(0) MSB	17153	255
S(1) LSB	17154	59
S(1) MSB	17155	35
S(2) LSB	17156	1
S(2) MSB	17157	255
S(3) LSB	17158	3
S(3) MSB	17159	3
S(4) LSB	17160	126
S(4) MSB	17161	254
S(5) LSB	17162	128
S(5) MSB	17163	56
S(6) LSB	17164	6
S(6) MSB	17165	47
S(7) LSB	17166	203
S(7) MSB	17167	255
S(8) LSB	17168	203
S(8) MSB	17169	183
S(9) LSB	17170	119
S(9) MSB	17171	35
S(10) LSB	17172	11
S(10) MSB	17173	120
S(11) LSB	17174	177
S(11) MSB	17175	32
S(12) LSB	17176	239
S(12) MSB	17177	201

Fig. 2-5. A 13 element packed integer array.

string constants, like S$, are stored in the program itself and not in string space. Once the string constant has been packed with the machine language subroutine, the string will always hold the subroutine.

To find the starting address for the string, the VARPTR com-

mand must be used. In line 8, the variable I is set to VARPTR(S$). Although variable I doesn't point to the string's starting address, it does point to the following information:

LOCATION	CONTENTS
I	Length of the String
I+1	LSB of the String's Starting Address
I+2	MSB of the String's Starting Address

With this information, the string's starting address can be determined by

```
PEEK(I+2)*256+PEEK(I+1)
```

Now that the string's starting address can be determined, the string can be packed in much the same way as an array is packed. If you have entered the program in Listing 2-5, try running it. The program will break in line 9. The string should now be packed. Try listing the program. If your video display seemed to throw a fit, then the string is probably packed correctly. The reason for this strange listing is that the BASIC interpreter can't list a packed string correctly, and, therefore, will not print it correctly either. The packed string is displayed as control codes and BASIC keywords. If the string has been properly packed, you may delete lines 5-9 and lines 1000-1010. The reason for this is that once the string is packed it will always stay packed. Therefore, the string packing routine and the data statements are no longer needed by the program.

Listing 2-5

```
0 ' LISTING 2.5
1 S$=".........................."
5 ' POKE SUBROUTINE INTO S$
6 FORM=0TO25
7 READB
8 I=VARPTR(S$):SA=PEEK(I+2)*256+PEEK(I+1):POKESA+M,B
9 NEXTM:STOP
10 ' DISPLAY ALTERNATING LINES
15 CLS
20 FORY=0TO47STEP2
30 FORX=0TO127
40 SET(X,Y)
50 NEXTX
60 NEXTY
70 ' WAIT FOR SHORT DELAY
80 FORI=1TO100
90 NEXTI
100 ' SET STARTING ADDRESS
110 I=VARPTR(S$):POKE16526,PEEK(I+1):POKE16527,PEEK(I+2)
120 ' REVERSE THE GRAPHICS
```

```
130 I=USR(0)
140 ' WAIT FOR SHORT DELAY
150 FORI=1TO100
160 NEXTI
170 'KEEP DOING UNTIL BREAK PRESSED
180 GOTO80
1000 DATA33,255,59,35,1,255,3,3,126,254,128,56,6,47
1010 DATA203,255,203,183,119,35,11,120,177,32,239,201
```

Chapter 3
The Disk BASIC and DOS Links

Although the USR command is the simplest way to interface machine language with BASIC, there are other more advanced techniques that can be used. This chapter introduces a technique utilizing the Disk BASIC and the DOS links. By using these links, new commands can be added to Level II BASIC, and existing Level II commands can be enhanced.

LINKS TO BASIC

Figure 3-1 shows all of the Disk BASIC links. Figure 3-2 lists all of the DOS links. If Level II BASIC is initialized, the Disk BASIC links are set to jump to the L3 ERROR routine whenever a Disk BASIC command is encountered. If Disk BASIC is loaded, these jumps to the L3 ERROR routine are replaced with jumps to the appropriate Disk BASIC routines. The DOS links are set to RET instructions by the Level II BASIC initialization routine. If a DOS is not in memory, the DOS links will simply return to Level II BASIC whenever a DOS link is encountered. When a DOS is in memory, the DOS links are set to jump to DOS.

In order to make use of these links, the Level II BASIC interpreter must be understood. The BASIC interpreter uses register pair HL as a pointer to the current location in the BASIC program. Think of register pair HL as a program counter. Whenever one of these links is called, register pair HL will either point to the

Exit Location	BASIC Keyword	Exit Location	BASIC Keyword
4152H	CVI	417CH	FIELD
4155H	FN	417FH	GET
4158H	CVS	4182H	PUT
415BH	DEF	4185H	CLOSE
415EH	CVD	4188H	LOAD
4161H	EOF	418BH	MERGE
4164H	LOC	418EH	NAME
4167H	LOF	4191H	KILL
416AH	MKI$	4194H	&
416DH	MKS$	4197H	LSET
4170H	MKD$	419AH	RSET
4173H	CMD	419DH	INSTR
4176H	TIME$	41A0H	SAVE
4179H	OPEN	41A3H	LINE

Fig. 3-1. Disk BASIC links.

last byte executed or the next byte immediately following the BASIC keyword.

ENHANCING USR

Let's try using a DOS link to enhance the USR command. The link at 41A9H is called by the USR command just before the value to be passed is examined. By intercepting this link, the USR command can be enhanced to call subroutines without poking the starting address into memory. The format for this new USR command will be

USR starting address,(x)

Some examples of how this new USR command might be used are

```
10  N=USR27648,(25)
100 X2=USRN,(M1)
```

Listing 3-1 presents a program that utilizes the USR command to poke the starting address via the DOS link.

Exit Location	Call From ROM Location(s)	Exit Location	Call From ROM Location(s)
41A6H	19ECH	41C4H	0358H
41A9H	27FEH	41C7H	1EA6H
41ACH	1A1CH	41CAH	206FH
41AFH	0368H	41CDH	20C6H
41B2H	1AA1H	41DOH	2103H
41B5H	1AECH	41D3H	2108H and 2141H
41B8H	1AF2H	41D6H	219EH
41BBH	1B8CH and 1DB0H	41DCH	222DH
41BEH	2174H	41DFH	2278H and 2B44H
41C1H	032CH	41E2H	02B2H

First, the program intercepts the DOS link at 41A9H. It performs this task by placing a jump to the desired location and then returning to the Level II BASIC READY routine. Note that a JP 0072H will return to the BASIC READY routine. This jump should always be used whenever a machine language routine is to return to the Level II BASIC command mode. While there are other jumps which return to the Level II BASIC command mode, the jump at 0072H is the only one which is error free. Other jumps may cause a BASIC error.

Whenever a USR command is executed, Level II BASIC immediately calls the DOS link at 41A9H. Because this link has been intercepted, it will cause a jump to 7FCFH. Note that at this point the BASIC program pointer in register pair HL will be pointing to the USR keyword. The new USR routine will start by calling the ROM routine located at 2B01H. This routine performs the following

Bump the BASIC Program Pointer, and evaluate expression address 2B01H.

This routine will increment the value of the BASIC program pointer and will evaluate the expression.

The BASIC program pointer is bumped by RST 0010H until register pair HL points to the first character in the expression. The expression is then evaluated. On exit, register pair DE will hold the 16-bit result and register pair HL points to the character immediately following the expression.

So, after the call 2B01H, register pair DE will hold the starting address. This starting address is then stored in memory location 16526 (408EH). Remember that the normal USR command requires the starting address to be poked at memory locations 16526 and 16527. Thus, the routine has just poked the starting address automatically.

After poking the starting address, the program performs a syntax check. If a comma doesn't follow the expression which was evaluated, an SN ERROR will be issued. Next, the BASIC program pointer, in register pair HL, must be decremented. This is necessary for two reasons. First, the RST 0008H syntax check executes a RST 0010H immediately after the syntax check. Therefore, register pair HL will point to the left parenthesis. Remember that on entry to the DOS link, register pair HL pointed to the USR keyword. So, by decrementing register pair HL, the BASIC program pointer will be adjusted to the correct position which BASIC expects on return from the subroutine.

This procedure demonstrates an important point about the

BASIC program pointer. Whenever a Disk BASIC or DOS link is used, you must consider the consequences of adjusting the BASIC program pointer when writing a routine. If you don't plan ahead, your routine may cause SN errors everytime it is executed.

Listing 3-1

```
7FBB            00100           ORG     7FBBH
7FBB 3EC3       00110 START     LD      A,0C3H
7FBD 32A941     00120           LD      (41A9H),A
7FC0 21CF7F     00130           LD      HL,USR
7FC3 22AA41     00140           LD      (41AAH),HL
7FC6 21DA7F     00150           LD      HL,M1
7FC9 CD752B     00160           CALL    2B75H
7FCC C37200     00170           JP      0072H
7FCF CD012B     00180 USR       CALL    2B01H
7FD2 ED538E40   00190           LD      (408EH),DE
7FD6 CF         00200           RST     0008H
7FD7 2C         00210           DEFB    ','
7FD8 2B         00220           DEC     HL
7FD9 C9         00230           RET
7FDA 45         00240 M1        DEFM    'ENHANCED USR COMMAND'
7FDB 4E
7FDC 48
7FDD 41
7FDE 4E
7FDF 43
7FE0 45
7FE1 44
7FE2 20
7FE3 55
7FE4 53
7FE5 52
7FE6 20
7FE7 43
7FE8 4F
7FE9 4D
7FEA 4D
7FEB 41
7FEC 4E
7FED 44
7FEE 0D         00250           DEFB    13
7FEF 42         00260           DEFM    'BY MARK GOODWIN'
7FF0 59
7FF1 20
7FF2 4D
7FF3 41
7FF4 52
7FF5 4B
7FF6 20
7FF7 47
7FF8 4F
7FF9 4F
7FFA 44
7FFB 57
7FFC 49
7FFD 4E
7FFE 0D         00270           DEFB    13
7FFF 00         00280           DEFB    0
7FBB            00290           END     START
00000 TOTAL ERRORS

M1       7FDA
USR      7FCF
START    7FBB
```

Listing 3-1 Comments

100 - Set the starting address for the program.
110 - Load register A with a JP op code.
120 - Save the JP op code in register A at the first byte of the DOS link.
130 - Load register pair HL with the starting address for the new USR routine.
140 - Save the starting address for the new USR routine in register pair HL at the second and third bytes of the DOS link.
150 - Load register pair HL with the starting address for the message to be displayed.
160 - Go display the message.
170 - Jump to the Level II BASIC READY routine.
180 - Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the integer result in register pair DE.
190 - Save the subroutine's starting address in register pair DE.
200 - Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a comma.
210 - The character to be checked for is stored here.
220 - Decrement the current BASIC program pointer in register pair HL.
230 - Return to the normal Level II BASIC USR routine.
240 - Part of the sign-on message is stored here.
250 - This will cause a carriage return to be displayed as part of the sign-on message.
260 - Part of the sign-on message is stored here.
270 - This will cause a carriage return to be displayed as part of the sign-on message.
280 - The sign-on message terminator is stored here.
290 - End of the program.

USING THIS NEW COMMAND

The enhancement to the USR command can be tested with a machine language subroutine and a BASIC program. Listing 3-2 presents a short machine language subroutine suitable for this demonstration. Before writing a BASIC program to go along with this machine language subroutine, let's first consider what method will be used to interface this subroutine with a BASIC program. First of all, the machine language subroutine is short and doesn't have any zeros. Can you guess which method I chose for interfac-

Hex	Decimal
CD	205
7F	127
0A	10
29	41
C3	195
9A	154
0A	10

Fig. 3-3. Decimal equivalents for the object code in Listing 3-2.

ing? If you guessed string packing, you're absolutely right. In order to pack the subroutine into a string, we must first determine the object code's decimal equivalent. Figure 3-3 shows the decimal equivalent for the object code. Next, a BASIC program such as the one in Listing 3-3 might be used to pack the string.

Once the string is properly packed, delete lines 35-70 and lines 999-1000. The program should now appear as in Listing 3-4. Now run the program. As you can see, the machine language subroutine simply multiplies the value passed by two. Although this program demonstrates how the enhanced USR command is used, it really doesn't show it's advantages. First the enhanced USR command eliminates the need for having to poke the starting address into memory. This will cause easier program development and faster program execution. One added benefit is also caused by the elimination of the pokes. Elimination of the pokes will reduce memory usage.

Listing 3-2

```
0000 CD7F0A     00100 SUB       CALL      0A7FH
0003 29         00110           ADD       HL,HL
0004 C39A0A     00120           JP        0A9AH
0000            00130           END       SUB
00000 TOTAL ERRORS

SUB        0000
```

Listing 3-2 Comments

100 - Go get the value passed.

110 - Add the value passed in register pair HL to itself. The result in register pair HL will be the value passed times two.

120 - Return to Level II BASIC with the adjusted value in register pair HL.

130 - End of the program.

Listing 3-3

```
10 ' LISTING 3.3
20 ' ENHANCED USR DEMO (BASIC)
30 S$="......."
35 ' PACK THE SUBROUTINE INTO S$
40 FORM=0TO6
50 READB
60 I=VARPTR(S$):SA=PEEK(I+2)*256+PEEK(I+1):POKESA+M,B
70 NEXTM:STOP
80 CLS
90 FORN=1TO10
100 PRINTN;"* 2 =";
110 I=VARPTR(S$):N1=USRPEEK(I+2)*256+PEEK(I+1),(N)
120 PRINTN1
130 NEXTN
999 END
1000 DATA205,127,10,41,195,154,10
```

Listing 3-4

```
10 ' LISTING 3.4
20 ' ENHANCED USR DEMO (BASIC)
30 S$="+
)ERRDEFSNG
"
80 CLS
90 FORN=1TO10
100 PRINTN;"* 2 =";
110 I=VARPTR(S$):N1=USRPEEK(I+2)*256+PEEK(I+1),(N)
120 PRINTN1
130 NEXTN
```

Chapter 4
Below BASIC

Have you ever wondered how some machine language utilities, such as DOS or Disk BASIC, can operate between Level II BASIC and the BASIC program area? Figure 4-1 presents a memory map for Level II BASIC. Note that the BASIC program area immediately follows the Level II BASIC interpreter. Now examine the memory map in Fig. 4-2, which shows how memory would be configured if a DOS or Disk BASIC were present. A DOS or Disk BASIC is loaded into memory immediately following the Level II BASIC interpreter, and the BASIC program area follows the DOS or Disk BASIC in memory.

Wouldn't it be nice to be able to write machine language subroutines that could reside in this area of memory between the Level II BASIC interpreter and the BASIC program area? Well, it is really quite simple to do. All that has to be done is to tell the Level II BASIC interpreter that the BASIC program area will be located higher in memory than usual.

MOVING THE BASIC PROGRAM AREA

In order to make a program operate below the BASIC program area, Level II BASIC must be told where the new BASIC program area will reside. The obvious place for it is immediately following the machine language subroutine. The new start of the BASIC program area must be stored at memory location 40A4H. Memory

0000H	Level II Basic
42E8H	
42E9H	Program Area
	Simple Variables
	Array Variables
	Free Memory
	Stack
	String Space
	Reserved Memory (If any)
7FFFH	End of Memory - 16K
BFFFH	End of Memory - 32K
FFFFH	End of Memory - 48K

Fig. 4-1. Level II BASIC memory map.

location 40A4H is used by Level II BASIC to point to the first byte available for BASIC program material.

One more point must be taken into consideration. The byte immediately preceding the start of the BASIC program area must

0000H	Level II BASIC
42E8H	
42E9H	DOS
	Disk BASIC
	Program Area
	Simple Variables
	Array Variables
	Free Memory
	Stack
	String Space
	Reserved Memory (If any)
7FFFH	End of Memory - 16K
BFFFH	End of Memory - 32K
FFFFH	End of Memory - 48K

Fig. 4-2. Disk BASIC memory map.

always equal zero (00H). The following example can be used to meet the above conditions.

```
;INITIALIZE NEW START OF BASIC

            LD      HL,BASIC  ;POINT TO LAST BYTE
                    OF PROGRAM
            LD      (HL),0    ;ZERO THIS BYTE

            INC     HL        ;BUMP POINTER

            LD      (40A4H),HL;SAVE BASIC START

            .

            .

            .

BASIC       DEFB    0

            END
```

Next the string space pointer must be reset in order to assure proper Level II BASIC operation. The string space pointer can be reset by calling the ROM routine at 1E83H.

Reset String Space Pointer, address 1E83H. On entry to this routine, register pair DE holds the number of bytes to be reserved as string space. The routine will reserve the amount of string space specified and reset the string space pointer to it's proper position. Level II BASIC reserves 50 bytes of string space on initialization. The following example meets both of these criteria.

```
;RESET THE STRING SPACE POINTER

;AND RESERVE 50 BYTES FOR STRING SPACE

            LD      DE,50     ;NUMBER OF BYTES

            CALL    1E83H     ;GO TO ROUTINE
```

One last thing must be done for proper operation of Level II BASIC. The simple variables pointer, the array variables pointer, the free space pointer, and other miscellaneous pointers must be reset. This is easily accomplished by calling the ROM routine located at 1B4DH.

After these steps have been completed in their proper order, Level II BASIC will function properly with the machine language

subroutine imbedded between the Level II BASIC interpreter and the BASIC program area. The short program listing in Fig. 4-3 presents all steps necessary to write programs in this manner.

TABBING BEYOND 63

Before trying to write your own program using this method, it may be best to study the example presented in Listing 4-1. The

```
;INSERT YOUR OWN MATERIAL WHERE INDICATED
          ORG     42E9H
START     CALL    01C9H          ;CLEAR THE SCREEN
          LD      HL,M1          ;POINT TO MESSAGE
          CALL    2B75H          ;DISPLAY MESSAGE
          LD      HL,BASIC       ;POINT TO BASIC START
          LD      (HL),0         ;ZERO IT
          LD      (40A4H),HL     ;SAVE BASIC START
          LD      DE,50          ;# OF BYTES TO RESERVE
          CALL    1E83H          ;RESET STRING SPACE
          CALL    1B4DH          ;RESET BASIC POINTERS
;INSERT ANY DISK BASIC OR DOS LINKS HERE
          .
          .
          .
          JP      0072H          ;RETURN TO LEVEL II
;INSERT ANY PROGRAM MATERIAL HERE
          .
          .
M1        DEFM    'INSERT SIGN ON MESSAGE HERE'
          .
          .
          .
BASIC     DEFB    0
          END     START
```

Fig. 4-3. Below BASIC.

program enhances the Level II TAB command so that it will permit TABs greater than 63 on the printer. If your computer has the newer Level II BASIC ROMs, this TAB extender isn't necessary; but you should study the program to better understand this Below BASIC interfacing method.

The program starts by clearing the screen and displaying a sign-on message. Next, the program sets the start of the BASIC program area to the end of the TAB enhancer program. String space is reserved and the pointers are reset. Then, the program links with the TAB routine via the DOS link located at 41D3H. Once this link is set, program control is returned to Level II BASIC.

Now that the program is linked with Level II BASIC, every time a TAB statement is encountered, the Level II BASIC interpreter will branch to the machine language subroutine via the DOS link. The subroutine first checks to see if the current output device is the printer. This is accomplished by loading the contents of 409CH, the current output device flag, into register A. Next, register A is ORed with itself to set the flags. This causes a RET M if the cassette is the current output device and a RET Z if the video display is the current output device.

DOS link 41D3H is called from two locations. These locations are 2108H and 2141H. The TAB command calls this DOS link from 2141H. Therefore, the return address must be checked to insure that the DOS link was called from 2141H and not from 2108H. If the DOS link was called from 2108H, the return address on the stack will be 210BH. Note that the MSB for both calls is 21H. Therefore, it is only necessary to check the LSB of the return address. If the LSB of the return address is equal to OBH, the subroutine will return to the Level II BASIC routine which called the DOS link.

Once the return address has been checked for it's proper value, the subroutine discards the return address. The return address can be discarded because program control will be returned to the Level II BASIC interpreter by a JP 214EH. Next, the BASIC program pointer is taken off the stack and placed in register pair HL. On entry to the DOS link, it points past the TAB(N) statement. Therefore, it is necessary to back up the pointer to the TAB(keyword. Because BASIC stores keywords in memory as one byte values, the program searches for a byte equal to BCH. BCH is the value Level II BASIC gives to TAB(. Once the TAB(keyword is located, the subroutine evaluates the TAB position by calling the ROM routine located at 2B1BH and places the result in register A.

Evaluate Expression, address 2B1BH. This rou-

tine first bumps the BASIC program pointer to the first character of the expression via the RST 0010H. The expression is then evaluated and the 8-bit result is returned in register A. On exit, the BASIC program pointer in register pair HL, will point to the byte immediately following the expression.

The TAB is transferred from register A to register E because, upon return to Level II BASIC, the TAB(N) command will expect the TAB position to be in register E. Next a syntax check is made. Then, the BASIC program pointer is decremented and placed on the stack. Control is then returned to Level II BASIC, by a JP 214EH.

Program Listing 4-2 is a short BASIC program which will test the TAB enhancer program. Note that with the TAB enhancer program, Level II BASIC will do any TAB from 0 to 255 on a printer.

Listing 4-1

```
42E9                    ORG     42E9H
42E9 CDC901      START  CALL    01C9H
42EC 213343             LD      HL,M1
42EF CD752B             CALL    2B75H
42F2 215143             LD      HL,BASIC
42F5 3600               LD      (HL),0
42F7 23                 INC     HL
42F8 22A440             LD      (40A4H),HL
42FB 113200             LD      DE,50
42FE CD831E             CALL    1E83H
4301 CD4D1B             CALL    1B4DH
4304 3EC3               LD      A,0C3H
4306 32D341             LD      (41D3H),A
4309 211243             LD      HL,TAB
430C 22D441             LD      (41D4H),HL
430F C37200             JP      0072H
4312 3A9C40      TAB    LD      A,(409CH)
4315 B7                 OR      A
4316 F8                 RET     M
4317 C8                 RET     Z
4318 E3                 EX      (SP),HL
4319 3E0B               LD      A,0BH
431B BD                 CP      L
431C 2002               JR      NZ,TAB1
431E E3                 EX      (SP),HL
431F C9                 RET
4320 E1          TAB1   POP     HL
4321 E1                 POP     HL
4322 2B          TAB2   DEC     HL
4323 7E                 LD      A,(HL)
4324 FEBC               CP      0BCH
4326 20FA               JR      NZ,TAB2
4328 CD1B2B             CALL    2B1BH
432B 5F                 LD      E,A
432C CF                 RST     0008H
432D 29                 DEFB    ')'
432E 2B                 DEC     HL
432F E5                 PUSH    HL
4330 C34E21             JP      214EH
4333 54          M1     DEFM    'TAB EXTENDER'
```

```
4334 41
4335 42
4336 20
4337 45
4338 58
4339 54
433A 45
433B 4E
433C 44
433D 45
433E 52
433F 0D         00500         DEFB    13
4340 42         00510         DEFM    'BY MARK GOODWIN'
4341 59
4342 20
4343 4D
4344 41
4345 52
4346 4B
4347 20
4348 47
4349 4F
434A 4F
434B 44
434C 57
434D 49
434E 4E
434F 0D         00520         DEFB    13
4350 00         00530         DEFB    0
4351 00         00540 BASIC   DEFB    0
42E9            00550         END     START
00000 TOTAL ERRORS

TAB2     4322
TAB1     4320
TAB      4312
BASIC    4351
M1       4333
START    42E9
```

Listing 4-1 Comments

100 - Set the program's starting address to the normal start of the BASIC program area.

110 - Go clear the screen.

120 - Load register pair HL with the starting address for the message to be displayed.

130 - Go display the message.

140 - Load register pair HL with the end of the TAB enhancer program.

150 - Zero the address at the location of the start of the BASIC program area pointer in register pair HL.

160 - Bump the start of the BASIC program area pointer in register pair HL.

170 - Save the start of the BASIC program area pointer in register pair HL.

180 - Load register pair DE with the number of bytes to be reserved as string space.

190 - Go reserve string space and reset the string space pointer.
200 - Go reset the BASIC pointers and variables.
210 - Load register A with a JP op code.
220 - Save the JP op code in register A as the first byte of the DOS link.
230 - Load register pair HL with the starting address of the TAB enhancer program.
240 - Save the starting address of the TAB enhancer program in register pair HL as the second and third bytes of the DOS link.
250 - Jump to the Level II BASIC READY routine.
260 - Load register A with the current output device code.
270 - Set the flags according to the current output device code in register A.
280 - Return if the current output device is the cassette.
290 - Return if the current output device is the video display.
300 - Exchange the value in register pair HL with the return address on the stack.
310 - Load register A with the LSB of the return address to check for.
320 - Check to see if the return address to be checked for in register A is the same as the LSB of the return address in register L.
330 - Jump if the LSB of the return address in register L doesn't match the value in register A.
340 - Exchange the return address in register pair HL with value on the stack.
350 - Return.
360 - Get the value from the stack and put it in register pair HL.
370 - Get the correct BASIC program pointer from the stack and put in register pair HL.
380 - Decrement the current BASIC program pointer in register pair HL.
390 - Load register A with the character at the location of the current BASIC program pointer in register pair HL.
400 - Check to see if the character in register A is a TAB(keyword.
410 - Jump if the character in register A isn't a TAB(keyword.
420 - Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the 8-bit result in register A.
430 - Load register E with the TAB value in register A.

440 - Go check the syntax. The byte at the location of the current BASIC program pointer in register pair HL must be a).
450 - The character to be checked for is stored here.
460 - Decrement the current BASIC program pointer in register pair HL.
470 - Save the current BASIC program pointer in register pair HL on the stack.
480 - Jump to the Level II BASIC TAB routine.
490 - Part of the sign-on message is stored here.
500 - This will display a carriage return as part of the sign-on message.
510 - Part of the sign-on message is stored here.
520 - This will display a carriage return as part of the sign-on message.
530 - The sign-on message terminator is stored here.
540 - The new start of the BASIC program area will start here.
550 - End of the program.

Listing 4-2

```
10 ' LISTING 4.2
20 ' TAB EXTENDER DEMO
30 FORN=0TO131
40 LPRINTTAB(N)"*"
50 NEXTN
```

Chapter 5

Adding New BASIC Commands

Although the Disk BASIC links can be used to add new commands, the name of a new command is limited to the names for the existing Disk BASIC commands. Wouldn't it be nice to be able to use SOUND as the name for the new BASIC command which would send sound out the cassette port? This chapter presents a method that enables you to add new BASIC commands and give these new commands any name you desire. Although this method does have a few limitations, it is preferable to other methods for adding new BASIC commands.

INTERCEPTING ERRORS

This method uses a DOS link at 41A6H. This DOS link is called by the Level II BASIC error routine. It's primary function is to branch to Disk BASIC so that Disk BASIC can display spelled-out error messages. On entry to this DOS link, register E holds the error code. The routine must check this error code to see if a syntax error has occurred. Register E must be checked for the syntax error code because using a new name for a BASIC command which isn't part of the reserved words list will cause a syntax error. Try typing DSET(0,16) followed by pressing ENTER on your computer. You will get a syntax error everytime it is entered. Therefore, the program must check for a syntax error. If an error occurs and it isn't a syntax error, the program will have to return from the DOS link without any further processing. If a syntax error does occur, regis-

ter E will be equal to two. A routine which would handle the above check could be written like this.

```
            PUSH      AF         ;SAVE AF
            LD        A,E        ;PUT ERROR CODE IN A
            CP        2          ;CHECK FOR SN ERROR
            JR        NZ,NOSN    ;IF NOT THEN RETURN
            .
            .
            .
NOSN        POP       AF         ;GET AF
            RET                  ;RETURN FROM DOS LINK
```

Once it has been determined that a syntax error has occurred, the program must check the syntax error to see if it is one of the new commands. Memory location 40E6H contains the position of the BASIC program pointer just before the error occurred. The program pointer will point to one of the two possibilities as shown in Fig. 5-1.

As shown in Fig. 5-1, if the syntax error occurred at the start of the new line; memory location 40E6H will point to the end of the last BASIC line. Therefore, the value contained in memory location 40E6H must be incremented by four to point to the byte immediately preceding the error. If the error occurs inside of a BASIC program line, memory location 40E6H will already be pointing to the byte immediately preceding the error.

Once the error location has been determined, the syntax error must be checked to see if it matches a new command. If you were going to add just one new command, a simple byte by byte compari-

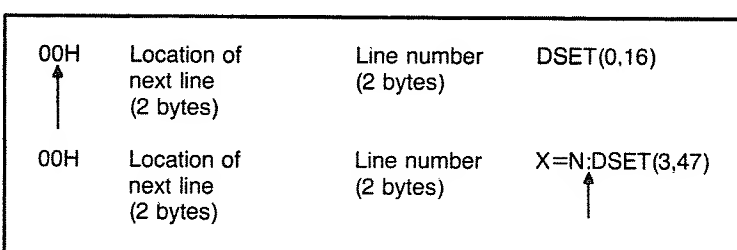

Fig. 5-1. BASIC program pointer positions.

son would be the easiest method to use for the syntax check. If more than one command is to be added, a table search would probably be the best method.

If the syntax error matches a new command, the program should jump to the proper function. Also, the program should set memory location 409AH to zero. Memory location 409AH is used by Level II BASIC as an error flag. To return to Level II BASIC after the error, the return address should be removed from the stack and a JP 1D1EH should be used.

If the syntax error doesn't match a new command, the program should return to the ERROR routine via the DOS link return address on the stack.

DOUBLE WIDTH GRAPHICS

The next program adds two new commands to Level II BASIC. These commands enable the use of double width graphics. The syntax for these two new commands and the new ROM routines are as follows.

DSET(X,Y). This command will set a double width graphics point on the video display at the coordinates specified by X and Y.

```
X must be>=0 and<=63.

Y must be>=0 and<=47.
```

For example 10 DSET(0,32) would be the same as
10 SET(0,32):SET(1,32).

DRESET(X,Y). This command will reset a double width graphics point on the video display at the coordinates specified by X and Y.

```
X must be>=0 and<=63

Y must be>=0 and<=47
```

For example, 10 DRESET(5,16) is the same as
10 RESET(10,16):RESET(11,16).

Evaluate Expression, address 2B1CH. This routine evaluates the expression at the current location of the BASIC program pointer in register pair HL. The 8-bit result is returned in register A. On exit, the BASIC program pointer will point to the byte immediately following the expression.

Check Syntax and Evaluate Expression, address 2B17H. This routine will first do a syntax check before

evaluating the expression. The expression must be preceded by a comma or a syntax error will result. The rest of the routine functions the same as the ROM routine at 2B1CH described above.

FC ERROR Routine, address 1E4AH. This routine will jump to the Level II BASIC error routine and output a FC ERROR message.

Graphics, address 0150H. This routine will perform the SET, RESET, and POINT graphic functions. On entry, the stack should have the following values:

1. The return address
2. The graphic flag—00H POINT, 01H RESET and 80H SET
3. The X value

Register A should hold the Y value. Register pair HL should point to a) character because this routine will perform a syntax check upon completion. Memory location 018DH is equal to a) character, so register pair HL should point to this location. The following example illustrates these points.

```
;SET(1,30)
          LD      HL,END      ;POINT TO RETURN ADDRESS
          PUSH    HL          ;PUT IT ON STACK
          LD      A,80H       ;A=SET MODE
          PUSH    AF          ;PUT IT ON STACK
          LD      A,1         ;A=X VALUE
          PUSH    AF          ;PUT IT ON STACK
          LD      A,30        ;A=Y VALUE
          LD      HL,018DH    ;HL=DUMMY PROGRAM POINTER
          JP      0150H       ;GO DO GRAPHICS
END                           ;RETURN HERE ON COMPLETION
```

The program in Listing 5-1 uses a table search for syntax checks. Note that the commands are not searched for a DSET and DRESET. This is because both new commands contain reserved words and the BASIC interpreter will store all keywords as one byte values. So whenever a new command uses a reserved word in

it's syntax, the one byte value must be compared for proper checking. Also, reserved words can't be used to start a new command name. Such words as SETS and PUTON would be illegal uses of reserved words.

As mentioned at the start of this chapter, there are a few limitations with this method. The reserved word limitation is one and the IF.THEN.ELSE structure is another. The newly created words cannot be used inside of the IF.THEN.ELSE statements as this example shows.

This would not work properly.

```
100 IFX=1THENDSET(1,0)ELSEDRESET(1,0)
```

But this would be alright.

```
100 IFX=1THEN110ELSE120
110 DSET(1,0):GOTO130
120 DRESET(1,0)
```

Finally, Listing 5-2 presents a program which will demonstrate the use of DSET and DRESET. Although Listing 5-1 only adds two new commands, it is easily modified to add more.

Listing 5-1

```
42E9          00100         ORG     42E9H
42E9 CDC901   00110 START   CALL    01C9H
42EC 21CA43   00120         LD      HL,M1
42EF CD752B   00130         CALL    2B75H
42F2 21F143   00140         LD      HL,BASIC
42F5 3600     00150         LD      (HL),0
42F7 23       00160         INC     HL
42F8 22A440   00170         LD      (40A4H),HL
42FB 113200   00180         LD      DE,50
42FE CD831E   00190         CALL    1E83H
4301 CD4D1B   00200         CALL    1B4DH
4304 3EC3     00210         LD      A,0C3H
4306 32A641   00220         LD      (41A6H),A
4309 211243   00230         LD      HL,DOUBLE
430C 22A741   00240         LD      (41A7H),HL
430F C37200   00250         JP      0072H
4312 F5       00260 DOUBLE  PUSH    AF
4313 7B       00270         LD      A,E
4314 FE02     00280         CP      2
4316 2060     00290         JR      NZ,NOSN
4318 E5       00300         PUSH    HL
4319 D5       00310         PUSH    DE
431A 2AE640   00320         LD      HL,(40E6H)
431D AF       00330         XOR     A
431E BE       00340         CP      (HL)
431F 2004     00350         JR      NZ,DOU1
4321 23       00360         INC     HL
4322 23       00370         INC     HL
```

```
4323 23         00380        INC    HL
4324 23         00390        INC    HL
4325 D7         00400 DOU1   RST    0010H
4326 22C243     00410        LD     (V1),HL
4329 32C843     00420        LD     (V4),A
432C 216C43     00430        LD     HL,TABLE
432F 010A00     00440        LD     BC,10
4332 EDB1       00450 L1     CPIR
4334 2040       00460        JR     NZ,SNE
4336 22C443     00470        LD     (V2),HL
4339 ED43C643   00480        LD     (V3),BC
433D EB         00490        EX     DE,HL
433E 2AC243     00500        LD     HL,(V1)
4341 23         00510 L2     INC    HL
4342 1A         00520        LD     A,(DE)
4343 BE         00530        CP     (HL)
4344 2008       00540        JR     NZ,L3
4346 13         00550        INC    DE
4347 1A         00560        LD     A,(DE)
4348 FE00       00570        CP     0
434A 280E       00580        JR     Z,L4
434C 18F3       00590        JR     L2
434E 2AC443     00600 L3     LD     HL,(V2)
4351 ED4BC643   00610        LD     BC,(V3)
4355 3AC843     00620        LD     A,(V4)
4358 18D8       00630        JR     L1
435A 22E640     00640 L4     LD     (40E6H),HL
435D EB         00650        EX     DE,HL
435E 23         00660        INC    HL
435F 5E         00670        LD     E,(HL)
4360 23         00680        INC    HL
4361 56         00690        LD     D,(HL)
4362 EB         00700        EX     DE,HL
4363 F1         00710        POP    AF
4364 F1         00720        POP    AF
4365 F1         00730        POP    AF
4366 F1         00740        POP    AF
4367 AF         00750        XOR    A
4368 329A40     00760        LD     (409AH),A
436B E9         00770        JP     (HL)
436C 44         00780 TABLE  DEFB   'D'
436D 83         00790        DEFB   83H
436E 00         00800        DEFB   0
436F 7E43       00810        DEFW   DON
4371 44         00820        DEFM   'D'
4372 82         00830        DEFB   82H
4373 00         00840        DEFB   0
4374 7A43       00850        DEFW   DOFF
4376 D1         00860 SNE    POP    DE
4377 E1         00870        POP    HL
4378 F1         00880 NOSN   POP    AF
4379 C9         00890        RET
437A 3E01       00900 DOFF   LD     A,01H
437C 1802       00910        JR     GR3
437E 3E80       00920 DON    LD     A,80H
4380 32C943     00930 GR3    LD     (GFLAG),A
4383 2AE640     00940        LD     HL,(40E6H)
4386 D7         00950        RST    0010H
4387 CF         00960        RST    0008H
4388 28         00970        DEFB   '('
4389 CD1C2B     00980        CALL   2B1CH
438C FE40       00990        CP     64
438E D24A1E     01000        JP     NC,1E4AH
4391 F5         01010        PUSH   AF
4392 CD172B     01020        CALL   2B17H
4395 FE30       01030        CP     48
4397 D24A1E     01040        JP     NC,1E4AH
439A F5         01050        PUSH   AF
```

```
439B CD8C01          CALL    018CH
439E F1              POP     AF
439F 5F              LD      E,A
43A0 F1              POP     AF
43A1 87              ADD     A,A
43A2 57              LD      D,A
43A3 E5              PUSH    HL
43A4 D5              PUSH    DE
43A5 3AC943          LD      A,(GFLAG)
43A8 CDB943          CALL    GRAPH
43AB D1              POP     DE
43AC 14              INC     D
43AD 3AC943          LD      A,(GFLAG)
43B0 CDB943          CALL    GRAPH
43B3 E1              POP     HL
43B4 2B              DEC     HL
43B5 D7              RST     0010H
43B6 C31E1D          JP      1D1EH
43B9 F5      GRAPH   PUSH    AF
43BA D5              PUSH    DE
43BB 7B              LD      A,E
43BC 218D01          LD      HL,018DH
43BF C35001          JP      0150H
43C2 0000    V1      DEFW    0
43C4 0000    V2      DEFW    0
43C6 0000    V3      DEFW    0
43C8 00      V4      DEFB    0
43C9 00      GFLAG   DEFB    0
43CA 44      M1      DEFM    'DOUBLE WIDTH GRAPHICS'
43CB 4F
43CC 55
43CD 42
43CE 4C
43CF 45
43D0 20
43D1 57
43D2 49
43D3 44
43D4 54
43D5 48
43D6 20
43D7 47
43D8 52
43D9 41
43DA 50
43DB 48
43DC 49
43DD 43
43DE 53
43DF 0D              DEFB    13
43E0 42              DEFM    'BY MARK GOODWIN'
43E1 59
43E2 20
43E3 4D
43E4 41
43E5 52
43E6 4B
43E7 20
43E8 47
43E9 4F
43EA 4F
43EB 44
43EC 57
43ED 49
43EE 4E
43EF 0D              DEFB    13
43F0 00              DEFB    0
43F1 00      BASIC   DEFB    0
```

```
42E9            01400           END      START
00000 TOTAL ERRORS

GRAPH    43B9
GFLAG    43C9
GR3      4380
DOFF     437A
DON      437E
L4       435A
L3       434E
L2       4341
V3       43C6
V2       43C4
SNE      4376
L1       4332
TABLE    436C
V4       43C8
V1       43C2
DOU1     4325
NOSN     4378
DOUBLE   4312
BASIC    43F1
M1       43CA
START    42E9
```

Listing 5-1 Comments

100 - Set the starting address to the normal start of the BASIC program area.
110 - Go clear the screen.
120 - Load register pair HL with the starting address for the message to be displayed.
130 - Go display the message.
140 - Load register pair HL with the ending address of the program.
150 - Zero the location of the start of the BASIC program area in register pair HL.
160 - Bump the start of the BASIC program area pointer in register pair HL.
170 - Save the start of the BASIC program area pointer in register pair HL.
180 - Load register pair DE with the number of bytes to be reserved for string space.
190 - Go reserve string space and reset the string space pointer.
200 - Go reset the BASIC pointers and variables.
210 - Load register A with a JP op code.
220 - Save the JP op code in register A as the first byte of the DOS link.
230 - Load register pair HL with the starting address of the double width graphics routine.
240 - Save the starting address of the double width graphics

routine in register pair HL as the second and third bytes of the DOS link.

250 - Jump to the Level II BASIC READY routine.
260 - Save the value in register pair AF on the stack.
270 - Load register A with the error code in register E.
280 - Check to see if the error code in register A is a syntax error code.
290 - Jump if the error code in register A isn't a syntax error code.
300 - Save the value in register pair HL on the stack.
310 - Save the value in register pair DE on the stack.
320 - Load register pair HL with the syntax error location.
330 - Zero register A.
340 - Check to see if the character at the location of the current BASIC program pointer in register pair HL is equal to zero.
350 - Jump if the character at the location of the current BASIC program pointer in register pair HL isn't equal to zero.
360 - Bump the current BASIC program pointer in register pair HL.
370 - Bump the current BASIC program pointer in register pair HL.
380 - Bump the current BASIC program pointer in register pair HL.
390 - Bump the current BASIC program pointer in register pair HL.
400 - Go bump the current BASIC program pointer in register pair HL till it points to the next character.
410 - Save the current BASIC program pointer in register pair HL.
420 - Save the character at the location of the current BASIC program pointer in register A.
430 - Load register pair HL with the starting address of the command jump table.
440 - Load register pair BC with the length of the command jump table.
450 - Go search for the character in register A in the command jump table.
460 - Jump if the character in register A wasn't found in the command jump table.
470 - Save the current command jump table pointer in register pair HL.
480 - Save the remaining length of the command jump table in register pair BC.

490 - Load register pair DE with the current command jump table pointer in register pair HL.
500 - Load register pair HL with the current BASIC program pointer.
510 - Bump the current BASIC program pointer in register pair HL.
520 - Load register A with the character at the location of the current command jump table pointer in register pair DE.
530 - Check to see if the character at the location of the current BASIC program pointer in register pair HL matches the character at the location of the current jump table pointer in register A.
540 - Jump if the character at the location of the current BASIC program pointer in register pair HL doesn't match the character at the location of the current jump table pointer in register A.
550 - Bump the current command jump table pointer in register pair DE.
560 - Load register A with the character at the location of the command jump table pointer in register pair DE.
570 - Check to see if the character at the location of the command jump table pointer in register A is equal to zero.
580 - Jump if the character at the location of the command jump table pointer in register A is equal to zero.
590 - Continue comparison.
600 - Load register pair HL with the current command jump table pointer.
610 - Load register pair BC with the remaining length of the command jump table to be searched.
620 - Load register A with the character to be searched for.
630 - Jump.
640 - Save the current BASIC program pointer in register pair HL.
650 - Load register pair HL with the current command jump table pointer in register pair DE.
660 - Bump the current command jump table pointer in register pair HL.
670 - Load register E with the LSB of the jump address at the location of the command jump table pointer in register pair HL.
680 - Bump the command jump table pointer in register pair HL.
690 - Load register D with the MSB of the jump address at the

location of the command jump table pointer in register pair HL.

700 - Load register pair HL with the jump address in register pair DE.

710 - Clean up the stack.

720 - Clean up the stack.

730 - Clean up the stack.

740 - Clean up the stack.

750 - Zero register A.

760 - Clear the Level II BASIC error flag.

770 - Jump to the command's starting address in register pair HL.

780 - Part of the DSET command is stored here.

790 - Part of the DSET command is stored here.

800 - The DSET command terminator is stored here.

810 - The DSET command starting address is stored here.

820 - Part of the DRESET command is stored here.

830 - Part of the DRESET command is stored here.

840 - The DRESET command terminator is stored here.

850 - The DRESET command starting address is stored here.

860 - Get the value from the stack and put it in register pair DE.

870 - Get the value from the stack and put it in register pair HL.

880 - Get the value from the stack and put it in register pair AF.

890 - Return.

900 - Load register A with the RESET flag.

910 - Jump.

920 - Load register A with the SET flag.

930 - Save the graphic flag in register A.

940 - Load register pair HL with the current BASIC program pointer.

950 - Bump the current BASIC program pointer in register pair HL till it points to the next character.

960 - Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a (.

970 - The character to be checked for is stored here.

980 - Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the X value in register A.

990 - Check to see if the X value in register A is greater than 63.

1000 - Go to the Level II BASIC error routine and output a FC ERROR message, if the X value in register A is greater than 63.

1010 - Save the X value in register A on the stack.
1020 - Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the Y value in register A.
1030 - Check to see if the Y value in register A is greater than 47.
1040 - Go to the Level II BASIC error routine and output a FC ERROR message, if the Y value in register A is greater than 47.
1050 - Save the Y value in register A on the stack.
1060 - Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a).
1070 - Get the Y value from the stack and put it in register A.
1080 - Load register E with the Y value in register A.
1090 - Get the X value from the stack and put it in register A.
1100 - Multiply the X value in register A by two.
1110 - Load register D with the X value in register A.
1120 - Save the current BASIC program pointer in register pair HL on the stack.
1130 - Save the X and Y values in register pair DE on the stack.
1140 - Load register A with the graphic flag.
1150 - Go do the graphics.
1160 - Get the X and Y values from the stack and put it in register pair DE.
1170 - Bump the X value in register D.
1180 - Load register A with the graphic flag.
1190 - Go do the graphics.
1200 - Get the current BASIC program pointer from the stack and put it in register pair HL.
1210 - Decrement the current BASIC program pointer in register pair HL.
1220 - Bump the current BASIC program pointer in register pair HL till it points to the next character.
1230 - Jump to Level II BASIC.
1240 - Save the graphic flag in register A on the stack.
1250 - Save the X value in register D on the stack.
1260 - Load register A with the Y value in register E.
1270 - Load register pair HL with the dummy BASIC program pointer.
1280 - Jump to the Level II BASIC graphics routine.
1290 - A value from the command jump table search will be stored here.

1300 - A value from the command jump table search will be stored here.
1310 - A value from the command jump table search will be stored here.
1320 - A value from the command jump table search will be stored here.
1330 - The graphic flag will be stored here.
1340 - Part of the sign-on message is stored here.
1350 - This will display a carriage return as part of the sign-on message.
1360 - Part of the sign-on message is stored here.
1370 - This will display a carriage return as part of the sign-on message.
1380 - The sign-on message terminator is stored here.
1390 - The BASIC program area will start here.
1400 - End of the program.

Listing 5-2

```
0 ' LISTING 5.2
10 ' DOUBLE WIDTH GRAPHICS DEMO
15 CLS
20 FORX=0TO63
30 FORY=0TO47STEP2
40 DSET(X,Y)
50 NEXTY
60 NEXTX
70 FORX=0TO63
80 FORY=0TO47
90 IFPOINT(X*2,Y)THEN100ELSE110
100 DRESET(X,Y):GOTO120
110 DSET(X,Y)
120 NEXTY
130 NEXTX
140 GOTO70
```

Chapter 6
Programmable Shift Key Entries

On the Model I, the keyboard, the video display, and the printer are all controlled by device control blocks. These device control blocks are used by Level II BASIC to branch to the appropriate driver routines and to hold data relating to each device. The driver routine for each device is a program which handles the input or output for the device. This chapter describes how the keyboard driver can be intercepted to allow for modification of the driver routine.

THE KEYBOARD DEVICE CONTROL BLOCK

The keyboard device control block is shown in Fig. 6-1 as it appears in memory. The keyboard driver address is shown as 03E3H. By placing a new address in memory location 4016H, Level II BASIC will branch to a new driver routine. The next program presents a way to modify the keyboard driver to provide programmable shift key entries. Shift key entries provide for faster program development by allowing you to enter a string of characters simply by holding the shift key down and pressing any key from A to Z.

Figure 6-2 lists the shift key entries, as they are initialized by the program. The program also makes use of the following three Disk BASIC links:

> **LSET.** This command turns the shift key entries off thus allowing for normal keyboard operation. To use, simply type LSET and press ENTER.

```
4015H - Device type: 1
4016H - Driver address (with Level II BASIC this is 03E3H)
4018H - Not used
4019H - Not used
401AH - Not used
401BH - RAM Buffer Address
```

Fig. 6-1. Keyboard device control block.

RSET. This command turns the shift key entries on. To use, simply type RSET and press ENTER.

CMDletter=string. This command is used to change the value of a shift key entry. The shift key entry to

```
A - STRING$(        B - INT(
C - CHR$(           D - DATA
E - ELSE            F - LEFT$(
G - GOTO            H - RIGHT$(
I - INPUT           J - INKEY$
K - RUN ENTER       L - LIST
M - MID$(           N - NEXT
O - ASC(            P - LPRINT
Q - SYSTEM ENTER    R - RETURN
S - GOSUB           T - THEN
U - USING           V - VAL(
W - RND(            X - STR$(
Y - LEN(            Z - EDIT
```

Fig. 6-2. The shift key entries.

be changed is specified by 'letter' and must be any letter from A to Z. The value of the shift key entry will be changed to the string, which can be any Level II BASIC string. Some examples follow.

```
CMDA="AUTO"

CMDM=STRING$(31,191)
```

The string must not be any more than 32 characters in length. If the string is more than 32 characters in length, the shift key entry value will be set to the first 32 characters of the string.

THE SHIFT KEY PROGRAM

Let's examine the program in Listing 6-1. The program starts by moving the start of the BASIC program area above the shift key entry program. Next, the three Disk BASIC commands are linked with the program. The keyboard driver address is loaded into register pair HL and saved as part of the program. The RSET function is then called to initialize the shift key entries. Program control is then returned to Level II BASIC.

The LSET command turns off the shift key entries by first saving the BASIC program pointer. Then, the old keyboard driver address is retrieved and placed in the keyboard device control block. The BASIC program pointer is retrieved and program control is returned to Level II BASIC.

The RSET command turns on the shift key entries by first saving the BASIC program pointer. Then, the new keyboard driver address is stored in the keyboard device control block. The BASIC program pointer is retrieved and program control is returned to Level II BASIC.

CMD changes a shift key entry by first checking the character pointed to by register pair HL. If the character is less than an A or the character is greater than a Z, program control will be passed to the FC ERROR routine.

Next, 65 is subtracted from the character pointed to by register pair HL. This gives the program an offset of from 0 to 25. This value is then multiplied by 2 in order to give the program a two byte offset. This offset is placed in register pair DE.

The BASIC program pointer is saved on the stack and register pair HL is loaded with the location of the shift key entry table. Register pairs HL and DE are then added resulting in register pair HL pointing to the storage location of the shift key entry to be

changed. This address is loaded into register pair DE and saved. The BASIC program pointer is retrieved from the stack. Register pair HL is incremented until it points to the next character.

A syntax check is done to check for the = sign. Note that Level II BASIC treats = as a BASIC keyword and that all ='s are stored as D5H in memory. The number type flag is then set to a string. The string expression is evaluated by calling the ROM routine at 2337H. ROM routine 2337H performs as follows.

> **Evaluate Expression,** address 2337H. This routine evaluates the expression at the location of the current BASIC program pointer in register pair HL. The result is returned in REG1. On exit, the BASIC program pointer will be pointing to the byte immediately following the expression.

The number type flag is then checked to make sure that the expression was a string. RST 0020H is used to perform this function. This ROM routine functions as follows:

> **Test the Number Type Flag,** RST 0020H. This routine tests the number type flag to determine the current type of value in REG1. The flags register and register A will be as follows on exit:

	FLAGS	REGISTER A
INTEGER	NZ/C/M/E	−1
STRING	Z/C/P/E	0
SINGLE	NZ/C/P/O	1
DOUBLE	NZ/NC/P/E	5

Once this check is completed, program control is passed to the TM ERROR routine if the number type flag is any other value except a string. Once the string has been correctly evaluated, memory location 4121H will hold the VARPTR for the string result. The length of the string is checked to see if it is greater than 32. If the length of the string is greater than 32, the length will be set to 32. Otherwise, the length is set to the actual string length. Once the length of the string has been set, the string is block moved to the shift key entry location.

Listing 6-1

```
42E9             00100          ORG     42E9H
42E9  CDC901     00110  START   CALL    01C9H
42EC  214C47     00120          LD      HL,M1
42EF  CD752B     00130          CALL    2B75H
```

```
42F2 217B47    00140        LD     HL,BASIC
42F5 3600      00150        LD     (HL),0
42F7 23        00160        INC    HL
42F8 22A440    00170        LD     (40A4H),HL
42FB 113200    00180        LD     DE,50
42FE CD831E    00190        CALL   1E83H
4301 CD4D1B    00200        CALL   1B4DH
4304 3EC3      00210        LD     A,0C3H
4306 327341    00220        LD     (4173H),A
4309 213C43    00230        LD     HL,CMD
430C 227441    00240        LD     (4174H),HL
430F 329741    00250        LD     (4197H),A
4312 212D43    00260        LD     HL,LSET
4315 229841    00270        LD     (4198H),HL
4318 329A41    00280        LD     (419AH),A
431B 213343    00290        LD     HL,RSET
431E 229B41    00300        LD     (419BH),HL
4321 2A1640    00310        LD     HL,(4016H)
4324 22A043    00320        LD     (JUMP+1),HL
4327 CD3343    00330        CALL   RSET
432A C37200    00340        JP     0072H
432D E5        00350 LSET   PUSH   HL
432E 2AA043    00360        LD     HL,(JUMP+1)
4331 1804      00370        JR     LP1
4333 E5        00380 RSET   PUSH   HL
4334 218D43    00390        LD     HL,KEY
4337 221640    00400 LP1    LD     (4016H),HL
433A E1        00410        POP    HL
433B C9        00420        RET
433C FE41      00430 CMD    CP     65
433E DA4A1E    00440        JP     C,1E4AH
4341 FE5B      00450        CP     91
4343 D24A1E    00460        JP     NC,1E4AH
4346 D641      00470        SUB    65
4348 CB27      00480        SLA    A
434A 1600      00490        LD     D,0
434C 5F        00500        LD     E,A
434D E5        00510        PUSH   HL
434E 21B943    00520        LD     HL,TABLE
4351 19        00530        ADD    HL,DE
4352 5E        00540        LD     E,(HL)
4353 23        00550        INC    HL
4354 56        00560        LD     D,(HL)
4355 ED534747  00570        LD     (CM1),DE
4359 E1        00580        POP    HL
435A D7        00590        RST    0010H
435B CF        00600        RST    0008H
435C D5        00610        DEFB   0D5H
435D 3E03      00620        LD     A,3
435F 32AF40    00630        LD     (40AFH),A
4362 CD3723    00640        CALL   2337H
4365 22E640    00650        LD     (40E6H),HL
4368 E7        00660        RST    0020H
4369 C2F60A    00670        JP     NZ,0AF6H
436C 2A2141    00680        LD     HL,(4121H)
436F 7E        00690        LD     A,(HL)
4370 FE21      00700        CP     33
4372 3003      00710        JR     NC,CM2
4374 4F        00720        LD     C,A
4375 1802      00730        JR     CM3
4377 0E20      00740 CM2    LD     C,32
4379 0600      00750 CM3    LD     B,0
437B 23        00760        INC    HL
437C 5E        00770        LD     E,(HL)
437D 23        00780        INC    HL
437E 56        00790        LD     D,(HL)
437F EB        00800        EX     DE,HL
4380 ED5B4747  00810        LD     DE,(CM1)
```

```
4384 EDB0      00820        LDIR
4386 EB        00830        EX      DE,HL
4387 3600      00840        LD      (HL),0
4389 2AE640    00850        LD      HL,(40E6H)
438C C9        00860        RET
438D 3A4B47    00870 KEY    LD      A,(FLAG)
4390 B7        00880        OR      A
4391 280C      00890        JR      Z,JUMP
4393 2A4947    00900        LD      HL,(POINT)
4396 7E        00910 KEY1   LD      A,(HL)
4397 23        00920        INC     HL
4398 224947    00930        LD      (POINT),HL
439B 324B47    00940        LD      (FLAG),A
439E C9        00950        RET
439F CD0000    00960 JUMP   CALL    0000H
43A2 FE61      00970        CP      97
43A4 D8        00980        RET     C
43A5 FE80      00990        CP      128
43A7 D0        01000        RET     NC
43A8 D661      01010        SUB     97
43AA CB27      01020        SLA     A
43AC 1600      01030        LD      D,0
43AE 5F        01040        LD      E,A
43AF 21B943    01050        LD      HL,TABLE
43B2 19        01060        ADD     HL,DE
43B3 5E        01070        LD      E,(HL)
43B4 23        01080        INC     HL
43B5 56        01090        LD      D,(HL)
43B6 EB        01100        EX      DE,HL
43B7 18DD      01110        JR      KEY1
43B9 ED43      01120 TABLE  DEFW    SHA
43BB 0E44      01130        DEFW    SHB
43BD 2F44      01140        DEFW    SHC
43BF 5044      01150        DEFW    SHD
43C1 7144      01160        DEFW    SHE
43C3 9244      01170        DEFW    SHF
43C5 B344      01180        DEFW    SHG
43C7 D444      01190        DEFW    SHH
43C9 F544      01200        DEFW    SHI
43CB 1645      01210        DEFW    SHJ
43CD 3745      01220        DEFW    SHK
43CF 5845      01230        DEFW    SHL
43D1 7945      01240        DEFW    SHM
43D3 9A45      01250        DEFW    SHN
43D5 BB45      01260        DEFW    SHO
43D7 DC45      01270        DEFW    SHP
43D9 FD45      01280        DEFW    SHQ
43DB 1E46      01290        DEFW    SHR
43DD 3F46      01300        DEFW    SHS
43DF 6046      01310        DEFW    SHT
43E1 8146      01320        DEFW    SHU
43E3 A246      01330        DEFW    SHV
43E5 C346      01340        DEFW    SHW
43E7 E446      01350        DEFW    SHX
43E9 0547      01360        DEFW    SHY
43EB 2647      01370        DEFW    SHZ
43ED 53        01380 SHA    DEFM    'STRING$('
43EE 54
43EF 52
43F0 49
43F1 4E
43F2 47
43F3 24
43F4 28
43F5 00        01390        DEFB    0
0018           01400        DEFS    24
440E 49        01410 SHB    DEFM    'INT('
440F 4E
```

```
4410 54
4411 28
4412 00          01420           DEFB     0
001C            01430           DEFS     28
442F 43         01440 SHC       DEFM     'CHR$('
4430 48
4431 52
4432 24
4433 28
4434 00         01450           DEFB     0
001B            01460           DEFS     27
4450 44         01470 SHD       DEFM     'DATA'
4451 41
4452 54
4453 41
4454 00         01480           DEFB     0
001C            01490           DEFS     28
4471 45         01500 SHE       DEFM     'ELSE'
4472 4C
4473 53
4474 45
4475 00         01510           DEFB     0
001C            01520           DEFS     28
4492 4C         01530 SHF       DEFM     'LEFT$('
4493 45
4494 46
4495 54
4496 24
4497 28
4498 00         01540           DEFB     0
001A            01550           DEFS     26
44B3 47         01560 SHG       DEFM     'GOTO'
44B4 4F
44B5 54
44B6 4F
44B7 00         01570           DEFB     0
001C            01580           DEFS     28
44D4 52         01590 SHH       DEFM     'RIGHT$('
44D5 49
44D6 47
44D7 48
44D8 54
44D9 24
44DA 28
44DB 00         01600           DEFB     0
0019            01610           DEFS     25
44F5 49         01620 SHI       DEFM     'INPUT'
44F6 4E
44F7 50
44F8 55
44F9 54
44FA 00         01630           DEFB     0
001B            01640           DEFS     27
4516 49         01650 SHJ       DEFM     'INKEY$'
4517 4E
4518 4B
4519 45
451A 59
451B 24
451C 00         01660           DEFB     0
001A            01670           DEFS     26
4537 52         01680 SHK       DEFM     'RUN'
4538 55
4539 4E
453A 0D         01690           DEFB     13
453B 00         01700           DEFB     0
001C            01710           DEFS     28
```

```
4558 4C         01720 SHL      DEFM     'LIST'
4559 49
455A 53
455B 54
455C 00         01730          DEFB     0
001C            01740          DEFS     28
4579 4D         01750 SHM      DEFM     'MID$('
457A 49
457B 44
457C 24
457D 28
457E 00         01760          DEFB     0
001B            01770          DEFS     27
459A 4E         01780 SHN      DEFM     'NEXT'
459B 45
459C 58
459D 54
459E 00         01790          DEFB     0
001C            01800          DEFS     28
45BB 41         01810 SHO      DEFM     'ASC('
45BC 53
45BD 43
45BE 28
45BF 00         01820          DEFB     0
001C            01830          DEFS     28
45DC 4C         01840 SHP      DEFM     'LPRINT'
45DD 50
45DE 52
45DF 49
45E0 4E
45E1 54
45E2 00         01850          DEFB     0
001A            01860          DEFS     26
45FD 53         01870 SHQ      DEFM     'SYSTEM'
45FE 59
45FF 53
4600 54
4601 45
4602 4D
4603 0D         01880          DEFB     13
4604 00         01890          DEFB     0
0019            01900          DEFS     25
461E 52         01910 SHR      DEFM     'RETURN'
461F 45
4620 54
4621 55
4622 52
4623 4E
4624 00         01920          DEFB     0
001A            01930          DEFS     26
463F 47         01940 SHS      DEFM     'GOSUB'
4640 4F
4641 53
4642 55
4643 42
4644 00         01950          DEFB     0
001B            01960          DEFS     27
4660 54         01970 SHT      DEFM     'THEN'
4661 48
4662 45
4663 4E
4664 00         01980          DEFB     0
001C            01990          DEFS     28
4681 55         02000 SHU      DEFM     'USING'
4682 53
4683 49
4684 4E
4685 47
```

```
4686 00        02010        DEFB   0
001B           02020        DEFS   27
46A2 56        02030 SHV    DEFM   'VAL('
46A3 41
46A4 4C
46A5 28
46A6 00        02040        DEFB   0
001C           02050        DEFS   28
46C3 52        02060 SHW    DEFM   'RND('
46C4 4E
46C5 44
46C6 28
46C7 00        02070        DEFB   0
001C           02080        DEFS   28
46E4 53        02090 SHX    DEFM   'STR$('
46E5 54
46E6 52
46E7 24
46E8 28
46E9 00        02100        DEFB   0
001B           02110        DEFS   27
4705 4C        02120 SHY    DEFM   'LEN('
4706 45
4707 4E
4708 28
4709 00        02130        DEFB   0
001C           02140        DEFS   28
4726 45        02150 SHZ    DEFM   'EDIT'
4727 44
4728 49
4729 54
472A 00        02160        DEFB   0
001C           02170        DEFS   28
4747 0000      02180 CM1    DEFW   0
4749 0000      02190 POINT  DEFW   0
474B 00        02200 FLAG   DEFB   0
474C 50        02210 M1     DEFM   'PROGRAMABLE SHIFT
                                    KEY ENTRIES'
474D 52
474E 4F
474F 47
4750 52
4751 41
4752 4D
4753 41
4754 42
4755 4C
4756 45
4757 20
4758 53
4759 48
475A 49
475B 46
475C 54
475D 20
475E 4B
475F 45
4760 59
4761 20
4762 45
4763 4E
4764 54
4765 52
4766 49
4767 45
4768 53
4769 0D        02220        DEFB   13
476A 42        02230        DEFM   'BY MARK GOODWIN'
```

```
476B 59
476C 20
476D 4D
476E 41
476F 52
4770 4B
4771 20
4772 47
4773 4F
4774 4F
4775 44
4776 57
4777 49
4778 4E
4779 0D          02240          DEFB     13
477A 00          02250          DEFB     0
477B 00          02260 BASIC    DEFB     0
42E9             02270          END      START
00000 TOTAL ERRORS

SHZ       4726
SHY       4705
SHX       46E4
SHW       46C3
SHV       46A2
SHU       46B1
SHT       4660
SHS       463F
SHR       461E
SHQ       45FD
SHP       45DC
SHO       45BB
SHN       459A
SHM       4579
SHL       4558
SHK       4537
SHJ       4516
SHI       44F5
SHH       44D4
SHG       44B3
SHF       4492
SHE       4471
SHD       4450
SHC       442F
SHB       440E
SHA       43ED
KEY1      4396
POINT     4749
FLAG      474B
CM3       4379
CM2       4377
CM1       4747
TABLE     43B9
KEY       438D
LP1       4337
JUMP      439F
RSET      4333
LSET      432D
CMD       433C
BASIC     477B
M1        474C
START     42E9
```

Listing 6-1 Comments

100 - Set the starting address to the normal start of the BASIC program area.

110 - Go clear the screen.
120 - Load register pair HL with the starting address for the message to be displayed.
130 - Go display the message.
140 - Load register pair HL with the ending address of the program.
150 - Zero the location of the start of the BASIC program area pointer in register pair HL.
160 - Bump the start of the BASIC program pointer in register pair HL.
170 - Save the start of the BASIC program area pointer in register pair HL.
180 - Load register pair DE with the number of bytes to be reserved as string space.
190 - Go reserve the string space and reset the string space pointer.
200 - Go reset the BASIC pointers and variables.
210 - Load register A with a JP op code.
220 - Save the JP op code in register A as the first byte of the CMD Disk BASIC link.
230 - Load register pair HL with the starting address for the CMD routine.
240 - Save the starting address for the CMD routine in register pair HL as the second and third bytes of the CMD Disk BASIC link.
250 - Save the JP op code in register A as the first byte of the LSET Disk BASIC link.
260 - Load register pair HL with the starting address for the LSET routine.
270 - Save the starting address for the LSET routine in register pair HL as the second and third bytes of the LSET Disk BASIC link.
280 - Save the JP op code in register A as the first byte of the RSET Disk BASIC link.
290 - Load register pair HL with the starting address for the RSET routine.
300 - Save the starting address for the RSET routine in register pair HL as the second and third bytes of the RSET Disk BASIC link.
310 - Load register pair HL with the keyboard driver address.
320 - Save the old keyboard driver address in register pair HL.
330 - Go initialize the shift key entries.

340 - Jump to the Level II BASIC READY routine.
350 - Save the current BASIC program pointer in register pair HL on the stack.
360 - Load register pair HL with the old keyboard driver address.
370 - Jump.
380 - Save the current BASIC program pointer in register pair HL on the stack.
390 - Load register pair HL with the new keyboard driver address.
400 - Save the keyboard driver address in register pair HL.
410 - Get the current BASIC program pointer from the stack and put it in register pair HL.
420 - Return.
430 - Check to see if the character at the location of the current BASIC program pointer in register A is less than an A.
440 - Jump to the Level II BASIC error routine and output a FC ERROR message if the character at the location of the current BASIC program pointer in register A is less than an A.
450 - Check to see if the character at the location of the current BASIC program pointer in register A is greater than a Z.
460 - Jump to the Level II BASIC error routine and output a FC ERROR message if the character at the location of the current BASIC program pointer in register A is greater than a Z.
470 - Subtract 65 from the value in register A to give an offset of 0 to 25.
480 - Multiply the offset in register A by two.
490 - Zero register D.
500 - Load register E with the offset in register A.
510 - Save the current BASIC program pointer in register pair HL on the stack.
520 - Load register pair HL with the starting address of the shift key location table.
530 - Add the offset in register pair DE to the starting address of the shift key location table in register pair HL.
540 - Load register E with the LSB of the shift key's starting address at the location of the shift key location table pointer in register pair HL.
550 - Bump the shift key location table pointer in register pair HL.
560 - Load register D with the MSB of the shift key's starting

address at the location of the shift key location table pointer in register pair HL.

570 - Save the shift key's starting address in register pair DE.

580 - Get the current BASIC program pointer from the stack and put it in register pair HL.

590 - Go bump the current BASIC program pointer in register pair HL till it points to the next character.

600 - Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be an =.

610 - The character to be checked for is stored here.

620 - Load register A with the string number type code.

630 - Set the number type flag to string.

640 - Go evaluate the string expression at the location of the current BASIC program pointer in register pair HL and return with the result in REG1.

650 - Save the current BASIC program pointer in register pair HL.

660 - Go check the number type flag.

670 - Go to the Level II BASIC error routine and output a TM ERROR if the current value of the number type flag isn't a string.

680 - Load register pair HL with the string's VARPTR.

690 - Load register A with the length of the string at the location of the string's VARPTR in register pair HL.

700 - Check to see if the length of the string in register A is greater than 32.

710 - Jump if the length of the string in register A is greater than 32.

720 - Load register C with the length of the string in register A.

730 - Jump.

740 - Load register C with the length of the string.

750 - Zero register B.

760 - Bump the string's VARPTR in register pair HL.

770 - Load register E with the LSB of the string's starting address.

780 - Bump the string's VARPTR in register pair HL.

790 - Load register D with the MSB of the string's starting address.

800 - Load register pair HL with the string's starting address in register pair DE.

810 - Load register pair DE with the shift key's starting address.

820 - Move the string to the shift key's storage location.
830 - Load register pair HL with the end of the shift key's storage location.
840 - Zero the end of the shift key's storage location.
850 - Load register pair HL with the current BASIC program pointer.
860 - Return.
870 - Load register A with the shift key flag.
880 - Check to see if a shift key is still being processed.
890 - Jump if a shift key isn't being processed.
900 - Load register pair HL with the current shift key pointer.
910 - Load register A with the character at the location of the current shift key pointer in register pair HL.
920 - Bump the current shift key pointer in register pair HL.
930 - Save the current shift key pointer in register pair HL.
940 - Save the character in register A as the shift key flag.
950 - Return.
960 - Go call the old keyboard driver.
970 - Check to see if the key pressed is less than a shift A.
980 - Return if the key pressed in register A is less than a shift A.
990 - Check to see if the key pressed in register A is greater than a shift Z.
1000 - Return if the key pressed in register A is greater than a shift Z.
1010 - Subtract 97 from register A so that register A will hold an offset from 0 to 25.
1020 - Multiply the offset in register A by two.
1030 - Zero register D.
1040 - Load register E with the offset in register A.
1050 - Load register pair HL with the starting address of the shift key location table.
1060 - Add the offset in register pair DE to the starting address of the shift key location table in register pair HL.
1070 - Load register E with the LSB of the shift key's starting address at the location of the shift key location table pointer in register pair HL.
1080 - Bump the shift key location table pointer in register pair HL.
1090 - Load register D with the MSB of the shift key's starting address at the location of the shift key location table pointer in register pair HL.

1100 - Load register pair HL with the shift key's starting address in register pair DE.
1110 - Jump.
1120 - Shift A's starting address is stored here.
1130 - Shift B's starting address is stored here.
1140 - Shift C's starting address is stored here.
1150 - Shift D's starting address is stored here.
1160 - Shift E's starting address is stored here.
1170 - Shift F's starting address is stored here.
1180 - Shift G's starting address is stored here.
1190 - Shift H's starting address is stored here.
1200 - Shift I's starting address is stored here.
1210 - Shift J's starting address is stored here.
1220 - Shift K's starting address is stored here.
1230 - Shift L's starting address is stored here.
1240 - Shift M's starting address is stored here.
1250 - Shift N's starting address is stored here.
1260 - Shift O's starting address is stored here.
1270 - Shift P's starting address is stored here.
1280 - Shift Q's starting address is stored here.
1290 - Shift R's starting address is stored here.
1300 - Shift S's starting address is stored here.
1310 - Shift T's starting address is stored here.
1320 - Shift U's starting address is stored here.
1330 - Shift V's starting address is stored here.
1340 - Shift W's starting address is stored here.
1350 - Shift X's starting address is stored here.
1360 - Shift Y's starting address is stored here.
1370 - Shift Z's starting address is stored here.
1380 - Shift A is stored here.
1390 - Shift A's terminator is stored here.
1400 - Make sure there's enough room for 32 characters.
1410 - Shift B is stored here.
1420 - Shift B's terminator is stored here.
1430 - Make sure there's enough room for 32 characters.
1440 - Shift C is stored here.
1450 - Shift C's terminator is stored here.
1460 - Make sure there's enough room for 32 characters.
1470 - Shift D is stored here.
1480 - Shift D's terminator is stored here.
1490 - Make sure there's enough room for 32 characters.

1500 - Shift E is stored here.
1510 - Shift E's terminator is stored here.
1520 - Make sure there's enough room for 32 characters.
1530 - Shift F is stored here.
1540 - Shift F's terminator is stored here.
1550 - Make sure there's enough room for 32 characters.
1560 - Shift G is stored here.
1570 - Shift G's terminator is stored here.
1580 - Make sure there's enough room for 32 characters.
1590 - Shift H is stored here.
1600 - Shift H's terminator is stored here.
1610 - Make sure there's enough room for 32 characters.
1620 - Shift I is stored here.
1630 - Shift I's terminator is stored here.
1640 - Make sure there's enough room for 32 characters.
1650 - Shift J is stored here.
1660 - Shift J's terminator is stored here.
1670 - Make sure there's enough room for 32 characters.
1680 - Part of shift K is stored here.
1690 - This will cause a carriage return as part of shift K.
1700 - Shift K's terminator is stored here.
1710 - Make sure there's enough room for 32 characters.
1720 - Shift L is stored here.
1730 - Shift L's terminator is stored here.
1740 - Make sure there's enough room for 32 characters.
1750 - Shift M is stored here.
1760 - Shift M's terminator is stored here.
1770 - Make sure there's enough room for 32 characters.
1780 - Shift N is stored here.
1790 - Shift N's terminator is stored here.
1800 - Make sure there's enough room for 32 characters.
1810 - Shift O is stored here.
1820 - Shift O's terminator is stored here.
1830 - Make sure there's enough room for 32 characters.
1840 - Shift P is stored here.
1850 - Shift P's terminator is stored here.
1860 - Make sure there's enough room for 32 characters.
1870 - Part of shift Q is stored here.
1880 - This will cause a carriage return as part of shift Q.
1890 - Shift Q's terminator is stored here.
1900 - Make sure there's enough room for 32 characters.
1910 - Shift R is stored here.

1920 - Shift R's terminator is stored here.
1930 - Make sure there's enough room for 32 characters.
1940 - Shift S is stored here.
1950 - Shift S's terminator is stored here.
1960 - Make sure there's enough room for 32 characters.
1970 - Shift T is stored here.
1980 - Shift T's terminator is stored here.
1990 - Make sure there's enough room for 32 characters.
2000 - Shift U is stored here.
2010 - Shift U's terminator is stored here.
2020 - Make sure there's enough room for 32 characters.
2030 - Shift V is stored here.
2040 - Shift V's terminator is stored here.
2050 - Make sure there's enough room for 32 characters.
2060 - Shift W is stored here.
2070 - Shift W's terminator is stored here.
2080 - Make sure there's enough room for 32 characters.
2090 - Shift X is stored here.
2100 - Shift X's terminator is stored here.
2110 - Make sure there's enough room for 32 characters.
2120 - Shift Y is stored here.
2130 - Shift Y's terminator is stored here.
2140 - Make sure there's enough room for 32 characters.
2150 - Shift Z is stored here.
2160 - Shift Z's terminator is stored here.
2170 - Make sure there's enough room for 32 characters.
2180 - Temporary storage location.
2190 - The current shift key entry pointer is stored here.
2200 - The shift key entry flag is stored here.
2210 - Part of the sign-on message is stored here.
2220 - This will display a carriage return as part of the sign-on message.
2230 - Part of the sign-on message is stored here.
2240 - This will display a carriage return as part of the sign-on message.
2250 - The sign-on message terminator is stored here.
2260 - The start of the BASIC program area will start here.
2270 - End of the program.

THE NEW KEYBOARD DRIVER

The new keyboard driver first checks the shift key flag to see if there is a shift key entry to be processed. If the program is still

processing a shift key entry, the next character is placed in register A. The value in register A is returned to Level II BASIC.

If the shift key entry flag indicates there isn't a shift key entry to be processed, then the old keyboard driver routine is called. If the character returned by the old keyboard driver is less than a shift A or greater than a shift Z, the program will return to Level II BASIC.

If a shift A to shift Z was pressed, the driver program figures the address for the shift key entry. The first shift key character is placed in register A. The shift key entry flag is loaded with this character to indicate a shift key entry is being processed. Then, the character in register A is returned to Level II BASIC.

The value of the shift key entries can be easily changed before assembly to any initial values you desire.

Chapter 7
A RAM Printer Spooler

A RAM printer spooler can be added to Level II BASIC by simply intercepting the keyboard driver and the printer driver. For those who are not familiar with a printer spooler, a short explanation about spoolers is in order.

THE PURPOSE OF A SPOOLER

Under normal operation of Level II BASIC, program execution is temporarily suspended while characters are sent to the printer. By halting program execution, the computer's throughput is greatly reduced. After all, just think how much time would be saved if the computer didn't have to wait for the printer.

Well, a printer spooler will virtually eliminate these delays for the printer. There are two basic types of printer spoolers: the RAM spooler and the disk spooler. With both types of spoolers, the program checks to see if the printer is busy. If the printer is busy, the spooler program sends the character to be printed to a buffer area in RAM for the RAM printer spooler or to the disk for the disk spooler.

Let's look at the RAM printer spooler's operation in more detail. When the printer spooler is first initialized, you are asked to enter the size of the buffer area. This buffer area is used by the spooler program to temporarily hold the characters while the printer is busy. The size of the buffer area is determined by the memory size of the computer and the memory requirements of the

program which will be executed. Suppose the buffer area is set to 2000 bytes upon initialization of the spooler program. If a program of 1500 bytes was LLISTed, the spooler program will immediately send the whole program to the buffer area. There will be no noticeable delay while the spooler performs this operation. The Level II BASIC READY message will return almost instantly. Now, the operator is able to continue with his next task.

For instance, he might choose to load a new program or maybe run the program that is already in memory while the spooler program will be sending all of the characters to the printer one at a time until the buffer is empty.

You may be wondering what would happen if a program longer than 2000 bytes was LLISTed. The spooler program would start by sending the first 2000 bytes of the program to the buffer area. At this point, printer operation will function like a normal Level II printer operation. The spooler will take a character out of the buffer and send it to the printer. Also the spooler will place the next character sent to it in the buffer area. This taking one character out and putting another character in will continue until there aren't anymore characters left to be placed in the buffer area. At this time, the spooler will return control of the computer to the operator and continue sending characters to the printer until the buffer area is empty.

As you can see from the above examples, the larger the buffer area the fewer delays will occur. So when the spooler program is initialized the buffer area should be set to the largest practical size.

A disk spooler works in much the same way as the RAM spooler. Instead of sending the characters to a buffer area in RAM, they are stored on the disk while the printer is busy. Although the disk spooler doesn't use any of the computer's RAM, it is slower because of the time involved with disk read/writes. Therefore, the RAM spooler is the preferred method.

THE SPOOLER PROGRAM

Program Listing 7-1 is a RAM printer spooler. It uses the keyboard driver to determine if the printer is busy. If it is, the spooler program jumps to the normal keyboard driver routine. If the printer isn't busy, then the spooler program performs it's intended function.

The program also uses the printer driver to determine if a character is being sent to the printer or not. The printer driver is intercepted in much the same way as the keyboard driver is inter-

4025H	Device Type
4026H - 4027H	Driver Address
4028H	Lines per page
4029H	Lines printed so far
402AH	Not Used
402BH - 402CH	RAM Buffer Address

Fig. 7-1. The printer device control block.

cepted; a new driver address is put in the printer device control block. The printer device control block is shown in Fig. 7-1 as it appears in memory.

The program in Listing 7-1 uses only one new ROM routine which hasn't been previously discussed. This routine is located at 0A9DH. This routine simply sets the number type flag to integer.

Listing 7-1

```
42E9                 ORG    42E9H
42E9 CDC901    START CALL   01C9H
42EC 21AC43          LD     HL,M1
42EF CD752B          CALL   2B75H
42F2 CDB31B          CALL   1BB3H
42F5 D7              RST    0010H
42F6 CD5A1E          CALL   1E5AH
42F9 21E543          LD     HL,BASIC
42FC 22A643          LD     (SBUFF),HL
42FF 22A843          LD     (NBUFF),HL
4302 19              ADD    HL,DE
4303 2B              DEC    HL
4304 22AA43          LD     (EBUFF),HL
4307 23              INC    HL
4308 3600            LD     (HL),0
430A 23              INC    HL
430B 22A440          LD     (40A4H),HL
430E 113200          LD     DE,50
4311 CD831E          CALL   1E83H
4314 CD4D1B          CALL   1B4DH
4317 2A1640          LD     HL,(4016H)
431A 226A43          LD     (JMP1+1),HL
431D 213543          LD     HL,KEY
4320 221640          LD     (4016H),HL
4323 2A2640          LD     HL,(4026H)
4326 228643          LD     (JMP2+1),HL
4329 224943          LD     (JMP3+1),HL
432C 216C43          LD     HL,PRINT
432F 222640          LD     (4026H),HL
4332 C37200          JP     0072H
4335 2AA843    KEY   LD     HL,(NBUFF)
4338 ED5BA643        LD     DE,(SBUFF)
433C DF              RST    0018H
433D 282A            JR     Z,JMP1
433F CDD105          CALL   05D1H
4342 2025            JR     NZ,JMP1
4344 2AA643          LD     HL,(SBUFF)
```

```
4347 4E                      LD      C,(HL)
4348 CD0000          JMP3    CALL    0000H
434B CD9D0A                  CALL    0A9DH
434E 2AA643                  LD      HL,(SBUFF)
4351 ED5BA843                LD      DE,(NBUFF)
4355 CDC70B                  CALL    0BC7H
4358 E5                      PUSH    HL
4359 C1                      POP     BC
435A 2AA643                  LD      HL,(SBUFF)
435D E5                      PUSH    HL
435E D1                      POP     DE
435F 23                      INC     HL
4360 EDB0                    LDIR
4362 2AA843                  LD      HL,(NBUFF)
4365 2B                      DEC     HL
4366 22A843                  LD      (NBUFF),HL
4369 C30000          JMP1    JP      0000H
436C 2AA843          PRINT   LD      HL,(NBUFF)
436F ED5BAA43                LD      DE,(EBUFF)
4373 DF                      RST     0018H
4374 CA8043                  JP      Z,SUB2
4377 2AA843                  LD      HL,(NBUFF)
437A 71                      LD      (HL),C
437B 23                      INC     HL
437C 22A843                  LD      (NBUFF),HL
437F C9                      RET
4380 C5              SUB2    PUSH    BC
4381 2AA643                  LD      HL,(SBUFF)
4384 4E                      LD      C,(HL)
4385 CD0000          JMP2    CALL    0000H
4388 CD9D0A                  CALL    0A9DH
438B 2AA643                  LD      HL,(SBUFF)
438E ED5BA843                LD      DE,(NBUFF)
4392 CDC70B                  CALL    0BC7H
4395 E5                      PUSH    HL
4396 C1                      POP     BC
4397 2AA643                  LD      HL,(SBUFF)
439A E5                      PUSH    HL
439B D1                      POP     DE
439C 23                      INC     HL
439D EDB0                    LDIR
439F 2AA843                  LD      HL,(NBUFF)
43A2 2B                      DEC     HL
43A3 C1                      POP     BC
43A4 71                      LD      (HL),C
43A5 C9                      RET
43A6 0000            SBUFF   DEFW    0
43A8 0000            NBUFF   DEFW    0
43AA 0000            EBUFF   DEFW    0
43AC 52              M1      DEFM    'RAM PRINTER SPOOLER'
43AD 41
43AE 4D
43AF 20
43B0 50
43B1 52
43B2 49
43B3 4E
43B4 54
43B5 45
43B6 52
43B7 20
43B8 53
43B9 50
43BA 4F
43BB 4F
43BC 4C
43BD 45
43BE 52
```

```
43BF 0D          00970          DEFB     13
43C0 42          00980          DEFM     'BY MARK D. GOODWIN'
43C1 59
43C2 20
43C3 4D
43C4 41
43C5 52
43C6 4B
43C7 20
43C8 44
43C9 2E
43CA 20
43CB 47
43CC 4F
43CD 4F
43CE 44
43CF 57
43D0 49
43D1 4E
43D2 0D          00990          DEFB     13
43D3 45          01000          DEFM     'ENTER BUFFER SIZE'
43D4 4E
43D5 54
43D6 45
43D7 52
43D8 20
43D9 42
43DA 55
43DB 46
43DC 46
43DD 45
43DE 52
43DF 20
43E0 53
43E1 49
43E2 5A
43E3 45
43E4 00          01010          DEFB     0
43E5 00          01020 BASIC    DEFB     0
42E9             01030          END      START
00000 TOTAL ERRORS

SUB2     4380
PRINT    436C
JMP3     4348
JMP2     4385
KEY      4335
JMP1     4369
EBUFF    43AA
NBUFF    43A8
SBUFF    43A6
BASIC    43E5
M1       43AC
START    42E9
```

Listing 7-1 Comments

100 - Set the starting address for the program to the start of user RAM.

110 - Go clear the screen.

120 - Load register pair HL with the starting address for the message to be displayed.

130 - Go display the message.

140 - Go get the response from the keyboard.

150 - Bump the input buffer pointer in register pair HL till it points to the first character.

160 - Go evaluate the expression at the location of the input buffer pointer in register pair HL and return with the integer result in register pair DE. The value in register pair DE is the size of the spooler's buffer area.

170 - Load register pair HL with the end of the spooler program.

180 - Save the end of the spooler program in register pair HL as the start of the buffer area.

190 - Save the end of the spooler program in register pair HL as the next available location in the spooler buffer.

200 - Add the buffer size in register pair DE to the start of the buffer area in register pair HL.

210 - Decrement the end of the spooler buffer area pointer in register pair HL.

220 - Save the end of the spooler buffer area pointer in register pair HL.

230 - Bump the pointer in register pair HL so that it points to the new start of the BASIC program area.

240 - Zero the location of the start of the BASIC program area pointer in register pair HL.

250 - Bump the start of the BASIC program area pointer in register pair HL.

260 - Save the new start of the BASIC program area pointer in register pair HL.

270 - Load register pair DE with the number of bytes to be reserved as string space.

280 - Go reserve the amount of string space and reset the start of string space pointer.

290 - Go reset the Level II BASIC variables and pointers.

300 - Load register pair HL with the present keyboard driver address.

310 - Save the present keyboard driver address in register pair HL so that the program can jump to the old keyboard driver when necessary.

320 - Load register pair HL with the starting address of the new keyboard driver.

330 - Save the new keyboard driver address in register pair HL in the keyboard device control block.

340 - Load register pair HL with the present printer driver address.

350 - Save the present printer driver address in register pair HL so that the program can jump to the old printer driver when necessary.
360 - Save the present printer driver address in register pair HL so that the program can jump to the old printer driver when necessary.
370 - Load register pair HL with the starting address of the new printer driver.
380 - Save the new printer driver address in register pair HL in the printer device control block.
390 - Return to the Level II BASIC READY routine.
400 - Load register pair HL with the next available location in the spooler buffer area pointer.
410 - Load register pair DE with the start of the spooler buffer area pointer.
420 - Go compare the next available location in the spooler buffer area pointer in register pair HL to the start of the spooler buffer area pointer in register pair DE.
430 - If the next available location in the spooler buffer area pointer in register pair HL is the same as the start of the spooler buffer area pointer in register pair DE then jump to the normal Level II BASIC keyboard driver routine.
440 - Go check to see if the printer is busy.
450 - If the printer is busy, then jump to the normal Level II BASIC keyboard driver routine.
460 - Load register pair HL with the start of the spooler buffer area pointer.
470 - Load register C with the character at the location of the start of the spooler buffer area pointer in register pair HL.
480 - This is a dummy call, but will be set to call the normal Level II BASIC printer driver by the initialization phase of the spooler program.
490 - Go set the current number type flag to integer.
500 - Load register pair HL with the start of the spooler buffer area pointer.
510 - Load register pair DE with the next available location in the spooler buffer area pointer.
520 - Go subtract the start of the spooler buffer area pointer in register pair HL from the next available location in the spooler buffer area pointer in register pair DE.
530 - Save the length of the remaining characters in register pair HL on the stack.

540 - Get the length of the remaining characters from the stack and put it in register pair BC.
550 - Load register pair HL with the start of the spooler buffer area pointer.
560 - Save the start of the spooler buffer area pointer in register pair HL on the stack.
570 - Get the start of the spooler buffer area pointer in register pair HL.
580 - Bump the start of the spooler buffer area pointer in register pair HL.
590 - Move the remaining characters in the spooler buffer down one byte.
600 - Load register pair HL with the next available location in the spooler buffer area pointer.
610 - Decrement the next available location in the spooler buffer area pointer in register pair HL.
620 - Save the next available location in the spooler buffer area pointer in register pair HL.
630 - Dummy jump. This will be changed by the spooler program's initialization phase, so that it will jump to the normal Level II BASIC keyboard driver.
640 - Load register pair HL with the next available location in the spooler buffer area pointer.
650 - Load register pair DE with the end of the spooler buffer area pointer.
660 - Go compare the next available location in the spooler buffer area pointer in register pair HL to the end of the spooler buffer area pointer in register pair DE.
670 - Jump to the spooler 'one in, one out' routine, if the next available location in the spooler buffer area pointer in register pair HL is the same as the end of the spooler buffer area pointer in register pair DE.
680 - Load register pair HL with the next available location in the spooler buffer area pointer.
690 - Save the character to be sent to the printer at the location of the next available location in the spooler buffer area pointer in register pair HL.
700 - Bump the next available location in the spooler buffer area pointer in register pair HL.
710 - Save the next available location in the spooler buffer area pointer in register pair HL.

720 - Return.
730 - Save the character to be printed in register C on the stack.
740 - Load register pair HL with the start of the spooler buffer area pointer.
750 - Load register C with the next character to be sent to the printer at the location of the start of the spooler buffer area pointer in register pair HL.
760 - Dummy call. This will be changed by the spooler program's initialization phase, so that it will call the normal Level II BASIC printer driver routine.
770 - Go set the current number type flag to integer.
780 - Load register pair HL with the start of the spooler buffer area pointer.
790 - Load register pair DE with the next available location in the spooler buffer area pointer.
800 - Go subtract the start of the spooler buffer area pointer in register pair HL from the next available location in the spooler buffer area pointer in register pair DE.
810 - Save the length of the remaining characters in the spooler buffer area in register pair HL on the stack.
820 - Get the length of the remaining characters in the spooler buffer area from the stack and put it in register pair BC.
830 - Load register pair HL with the start of the spooler buffer area pointer.
840 - Save the start of the spooler buffer area pointer in register pair HL on the stack.
850 - Get the start of the spooler buffer area pointer from the stack and put it in register pair DE.
860 - Bump the start of the spooler buffer area pointer in register pair HL.
870 - Move the remaining characters in the spooler buffer area down one byte.
880 - Load register pair HL with the next available location in the spooler buffer area pointer.
890 - Decrement the next available location in the spooler buffer area pointer in register pair HL.
900 - Get the character to be put in the spooler buffer area from the stack and put it in register C.
910 - Save the character to be put in the spooler buffer area in register C at the location of the next available location of the spooler buffer area pointer in register pair HL.

920 - Return.
930 - Reserve two bytes for the start of the spooler buffer area pointer.
940 - Reserve two bytes for the next available location in the spooler buffer area pointer.
950 - Reserve two bytes for the end of the spooler buffer area pointer.
960 - Part of the sign-on message is stored here.
970 - This will cause a carriage return to be displayed as part of the sign-on message.
980 - Part of the sign-on message is stored here.
990 - This will cause a carriage return to be displayed as part of the sign-on message.
1000 - Part of the sign-on message is stored here.
1010 - The sign-on message terminator is stored here.
1020 - The spooler buffer area will start here.
1030 - End of the program.

Chapter 8

The JKL-to-Printer Function

The utility program in this chapter is called the JKL-TO-PRINTER function. This utility is included on many disk operating systems. The utility's function is quite simple. Whenever the JKL keys are all pressed at the same time, the utility will print the contents of the video display on the printer. It can be quite useful when debugging programs. If an error occurs, the operator can dump the contents of the screen on the printer, and this screen dump can be referred to later when trying to determine the problem in the program.

THE JKL-TO-PRINTER PROGRAM

The keyboard on the TRS-80 Model I is a memory-mapped device. Simply stated, this means that Level II BASIC looks at a specific location in memory to determine if a key is being pressed. The J, K, and L keys are all located at memory location 3802H. If all of the keys are being pressed at the same time, memory location 3802H will have the value of 28. Therefore, the program simply reads the value stored at 3802H. If it is equal to 28, the program knows that the JKL keys are all being pressed at the same time.

Program Listing 8-1 performs the JKL-TO-PRINTER function. The program intercepts the keyboard driver, upon initialization. Once initialized, the program will check to see if the JKL keys are being pressed all at the same time. This check is performed each time level II BASIC scans the keyboard. If the JKL keys are being held down, the program will dump the contents of the screen on the

printer. If the JKL keys aren't being held down, the program will jump to the normal Level II BASIC keyboard driver.

Listing 8-1

```
42E9                     ORG     42E9H
42E9 CDC901     START    CALL    01C9H
42EC 214D43              LD      HL,M1
42EF CD752B              CALL    2B75H
42F2 CD4900              CALL    0049H
42F5 FE59                CP      'Y'
42F7 2808                JR      Z,LP1
42F9 FE4E                CP      'N'
42FB 20EC                JR      NZ,START
42FD 3E3E                LD      A,3EH
42FF 1802                JR      LP2
4301 3EC6       LP1      LD      A,0C6H
4303 323B43     LP2      LD      (JKL2),A
4306 219243              LD      HL,BASIC
4309 3600                LD      (HL),0
430B 23                  INC     HL
430C 22A440              LD      (40A4H),HL
430F 113200              LD      DE,50
4312 CD831E              CALL    1E83H
4315 CD4D1B              CALL    1B4DH
4318 2A1640              LD      HL,(4016H)
431B 222D43              LD      (JUMP+1),HL
431E 212743              LD      HL,JKL
4321 221640              LD      (4016H),HL
4324 C37200              JP      0072H
4327 3A0238     JKL      LD      A,(3802H)
432A FE1C                CP      28
432C C20000     JUMP     JP      NZ,0000H
432F 21003C              LD      HL,3C00H
4332 0E10                LD      C,16
4334 0640       JKL3     LD      B,64
4336 7E         JKL4     LD      A,(HL)
4337 FE80                CP      128
4339 3802                JR      C,JKL5
433B 00         JKL2     NOP
433C 20                  DEFB    32
433D CD3B00     JKL5     CALL    003BH
4340 23                  INC     HL
4341 10F3                DJNZ    JKL4
4343 3E0D                LD      A,13
4345 CD3B00              CALL    003BH
4348 0D                  DEC     C
4349 20E9                JR      NZ,JKL3
434B AF                  XOR     A
434C C9                  RET
434D 4A         M1       DEFM    'JKL-TO-PRINTER'
434E 4B
434F 4C
4350 2D
4351 54
4352 4F
4353 2D
4354 50
4355 52
4356 49
4357 4E
4358 54
4359 45
435A 52
435B 0D                  DEFB    13
```

```
435C 42         00570           DEFM    'BY MARK D. GOODWIN'
435D 59
435E 20
435F 4D
4360 41
4361 52
4362 4B
4363 20
4364 44
4365 2E
4366 20
4367 47
4368 4F
4369 4F
436A 44
436B 57
436C 49
436D 4E
436E 0D         00580           DEFB 13
436F 44         00590           DEFM 'DO YOU HAVE AN EPSON MX-80 (Y/N)?'
4370 4F
4371 20
4372 59
4373 4F
4374 55
4375 20
4376 48
4377 41
4378 56
4379 45
437A 20
437B 41
437C 4E
437D 20
437E 45
437F 50
4380 53
4381 4F
4382 4E
4383 20
4384 4D
4385 58
4386 2D
4387 38
4388 30
4389 20
438A 28
438B 59
438C 2F
438D 4E
438E 29
438F 3F
4390 0D         00600           DEFB  13
4391 00         00610           DEFB  0
4392 00         00620 BASIC     DEFB  0
42E9            00630           END   START
00000 TOTAL ERRORS
JKL5     433D
JKL4     4336
JKL3     4334
JKL      4327
JUMP     432C
BASIC    4392
JKL2     433B
LP2      4303
LP1      4301
M1       434D
START    42E9
```

Listing 8-1 Comments

100 - Set the starting address of the program to the start of user RAM.
110 - Go clear the screen.
120 - Load register pair HL with the starting address for the message to be displayed.
130 - Go display the message.
140 - Go scan the keyboard and wait till a key has been pressed. Return with the key pressed in register A.
150 - Check to see if the key that was pressed in register A is a Y.
160 - Jump if the key that was pressed in register A is a Y.
170 - Check to see if the key that was pressed in register A is a N.
180 - Go ask again if the key that was pressed in register A isn't a N.
190 - Load register A with the op code for a LD A,N instruction.
200 - Jump.
210 - Load register A with the op code for a ADD A,N instruction.
220 - Save the op code in register A so that it will perform the proper operation when the JKL-to-printer function is used.
230 - Load register pair HL with the start of the BASIC program area pointer.
240 - Zero the location of the start of the BASIC program pointer in register pair HL.
250 - Bump the start of the BASIC program pointer in register pair HL.
260 - Save the start of the BASIC program area pointer in register pair HL.
270 - Load register pair DE with the number of bytes to be reserved as string space.
280 - Go reserve the amount of string space in register pair DE.
290 - Go reset the BASIC variables and pointers.
300 - Load register pair HL with the starting address of the present Level II BASIC keyboard driver.
310 - Save the starting address of the present Level II BASIC keyboard driver in register pair HL so that it can be used by the JKL-TO-PRINTER function to branch to the normal keyboard driver routine.
320 - Load register pair HL with the starting address of the JKL-TO-PRINTER routine.
330 - Save the starting address of the JKL-TO-PRINTER routine in register pair HL in the keyboard device control block as the current keyboard driver address.

340 - Jump to the Level II BASIC READY routine.
350 - Load register A with the memory location which holds the value for the JKL keys.
360 - Check to see if the value in register A is the same as the expected value for the JKL keys all being pressed at the same time.
370 - Dummy jump. This will be set to jump to the normal keyboard driver routine by the initialization process, if the JKL keys aren't being pressed all at the same time.
380 - Load register pair HL with the start of video memory.
390 - Load register C with the number of lines to be printed.
400 - Load register B with the number of characters per line to be printed.
410 - Load register A with the character being displayed at the location of the video memory pointer in register pair HL.
420 - Check to see if the video character in register A is a graphic character.
430 - If the video character in register A isn't a graphic character then jump.
440 - Dummy op code. This will be changed to either a LD A,N or a ADD A,N op code by the initialization process.
450 - The value to either load register A with or the value to add to register A is stored here. This is determined by the value of the op code in the previous line.
460 - Go print the character in register A on the printer.
470 - Bump the video memory pointer in register pair HL.
480 - Loop if all of the characters on the present line of the video display haven't been printed.
490 - Load register A with the ASCII code for a carriage return.
500 - Go print a carriage return on the printer.
510 - Decrement the number of lines to be printed in register C.
520 - Loop till all of the lines on the video display have been printed.
530 - Zero register A.
540 - Return.
550 - Part of the sign-on message is stored here.
560 - This will cause a carriage return to be displayed as part of the sign-on message.
570 - Part of the sign-on message is stored here.
580 - This will cause a carriage return to be displayed as part of the sign-on message.
590 - Part of the sign-on message is stored here.

600 - This will cause a carriage return to be displayed as part of the sign-on message.
610 - The sign-on message terminator is stored here.
620 - The new start of the BASIC program area will be here.
630 - End of the program.

EPSON MX-80 PRINTER

Most of the recent printers introduced on the market have graphics printing capability. The Epson MX-80 is one of them. The program presented in this chapter starts it's initialization phase by asking the operator if he has an Epson MX-80. If one is available, the JKL utility will be able to print any graphic characters displayed on the screen. The Epson graphic codes are the same as the Model I's, except that 32 must be added to the value of the Model I graphic code to print the same graphic character on the MX-80. Therefore, the program checks each character to be sent to the printer to see if it's a graphic character. If it is, the utility adds 32 to the character's value.

NON-GRAPHIC PRINTERS

If the printer available doesn't have graphics capability, then the appropriate response should be given to the program's initialization phase. By telling the JKL utility that an Epson MX-80 isn't available, the program will print a space for any graphic character appearing on the video display. The program performs this by checking each character to be sent to the printer to see whether or not it is a graphic character.

Chapter 9
Spelled-Out Error Messages

The abbreviated Level II BASIC error messages just don't provide enough information to be adequate. They're okay for errors which occur frequently, but a lot of time can be wasted looking up error messages which don't occur frequently.

The program in Listing 9-1 adds spelled-out error messages to Level II BASIC. Upon initialization of the program, it intercepts the DOS link at 41A6H. Whenever an error occurs in Level II BASIC, the error routine will call the DOS link. On entry to the DOS link, register E will hold the appropriate code. The program adds the value of the error code to a table pointer, which in turn, will point to the new error message to be displayed. The routine jumps back to Level II BASIC with the starting address of the new error message in register pair HL and the Level II BASIC error routine continues on normally.

Listing 9-1

```
42E9                00100           ORG     42E9H
42E9 CDC901         00110 START     CALL    01C9H
42EC 21E244         00120           LD      HL,M1
42EF CD752B         00130           CALL    2B75H
42F2 211145         00140           LD      HL,BASIC
42F5 3600           00150           LD      (HL),0
42F7 23             00160           INC     HL
42F8 22A440         00170           LD      (40A4H),HL
42FB 113200         00180           LD      DE,50
42FE CD831E         00190           CALL    1E83H
4301 CD4D1B         00200           CALL    1B4DH
```

```
4304 3EC3      00210          LD     A,0C3H
4306 32A641    00220          LD     (41A6H),A
4309 211243    00230          LD     HL,ERROR
430C 22A741    00240          LD     (41A7H),HL
430F C37200    00250          JP     0072H
4312 1600      00260 ERROR    LD     D,0
4314 211F43    00270          LD     HL,TABLE
4317 19        00280          ADD    HL,DE
4318 5E        00290          LD     E,(HL)
4319 23        00300          INC    HL
431A 56        00310          LD     D,(HL)
431B EB        00320          EX     DE,HL
431C C3011A    00330          JP     1A01H
431F 4D43      00340 TABLE    DEFW   NF
4321 5E43      00350          DEFW   SN
4323 6B43      00360          DEFW   RG
4325 8043      00370          DEFW   OD
4327 8C43      00380          DEFW   FC
4329 A243      00390          DEFW   OV
432B AB43      00400          DEFW   OM
432D B943      00410          DEFW   UL
432F CF43      00420          DEFW   BS
4331 E643      00430          DEFW   DD
4333 FA43      00440          DEFW   DO
4335 0B44      00450          DEFW   ID
4337 2444      00460          DEFW   TM
4339 3244      00470          DEFW   OS
433B 4644      00480          DEFW   LS
433D 5644      00490          DEFW   ST
433F 7144      00500          DEFW   CN
4341 8044      00510          DEFW   NR
4343 8A44      00520          DEFW   RW
4345 9F44      00530          DEFW   UE
4347 B144      00540          DEFW   MO
4349 C144      00550          DEFW   FD
434B CF44      00560          DEFW   L3
434D 4E        00570 NF       DEFM   'NEXT WITHOUT FOR'
434E 45
434F 58
4350 54
4351 20
4352 57
4353 49
4354 54
4355 48
4356 4F
4357 55
4358 54
4359 20
435A 46
435B 4F
435C 52
435D 00        00580          DEFB   0
435E 53        00590 SN       DEFM   'SYNTAX ERROR'
435F 59
4360 4E
4361 54
4362 41
4363 58
4364 20
4365 45
4366 52
4367 52
4368 4F
4369 52
436A 00        00600          DEFB   0
436B 52        00610 RG       DEFM   'RETURN WITHOUT GOSUB'
436C 45
```

```
436D 54
436E 55
436F 52
4370 4E
4371 20
4372 57
4373 49
4374 54
4375 48
4376 4F
4377 55
4378 54
4379 20
437A 47
437B 4F
437C 53
437D 55
437E 42
437F 00          00620          DEFB     0
4380 4F          00630 OD       DEFM     'OUT OF DATA'
4381 55
4382 54
4383 20
4384 4F
4385 46
4386 20
4387 44
4388 41
4389 54
438A 41
438B 00          00640          DEFB     0
438C 49          00650 FC       DEFM     'ILLEGAL FUNCTION CALL'
438D 4C
438E 4C
438F 45
4390 47
4391 41
4392 4C
4393 20
4394 46
4395 55
4396 4E
4397 43
4398 54
4399 49
439A 4F
439B 4E
439C 20
439D 43
439E 41
439F 4C
43A0 4C
43A1 00          00660          DEFB     0
43A2 4F          00670 OV       DEFM     'OVERFLOW'
43A3 56
43A4 45
43A5 52
43A6 46
43A7 4C
43A8 4F
43A9 57
43AA 00          00680          DEFB     0
43AB 4F          00690 OM       DEFM     'OUT OF MEMORY'
43AC 55
43AD 54
43AE 20
43AF 4F
43B0 46
```

```
43B1 20
43B2 4D
43B3 45
43B4 4D
43B5 4F
43B6 52
43B7 59
43B8 00        00700          DEFB     0
43B9 55        00710 UL       DEFM     'UNDEFINED LINE NUMBER'
43BA 4E
43BB 44
43BC 45
43BD 46
43BE 49
43BF 4E
43C0 45
43C1 44
43C2 20
43C3 4C
43C4 49
43C5 4E
43C6 45
43C7 20
43C8 4E
43C9 55
43CA 4D
43CB 42
43CC 45
43CD 52
43CE 00        00720          DEFB     0
43CF 53        00730 BS       DEFM     'SUBSCRIPT OUT OF RANGE'
43D0 55
43D1 42
43D2 53
43D3 43
43D4 52
43D5 49
43D6 50
43D7 54
43D8 20
43D9 4F
43DA 55
43DB 54
43DC 20
43DD 4F
43DE 46
43DF 20
43E0 52
43E1 41
43E2 4E
43E3 47
43E4 45
43E5 00        00740          DEFB     0
43E6 52        00750 DD       DEFM     'REDIMENSIONED ARRAY'
43E7 45
43E8 44
43E9 49
43EA 4D
43EB 45
43EC 4E
43ED 53
43EE 49
43EF 4F
43F0 4E
43F1 45
43F2 44
43F3 20
43F4 41
```

```
43F5 52
43F6 52
43F7 41
43F8 59
43F9 00        00760         DEFB     0
43FA 44        00770 DO      DEFM     'DIVISION BY ZERO'
43FB 49
43FC 56
43FD 49
43FE 53
43FF 49
4400 4F
4401 4E
4402 20
4403 42
4404 59
4405 20
4406 5A
4407 45
4408 52
4409 4F
440A 00        00780         DEFB     0
440B 49        00790 ID      DEFM     'ILLEGAL DIRECT OPERATION'
440C 4C
440D 4C
440E 45
440F 47
4410 41
4411 4C
4412 20
4413 44
4414 49
4415 52
4416 45
4417 43
4418 54
4419 20
441A 4F
441B 50
441C 45
441D 52
441E 41
441F 54
4420 49
4421 4F
4422 4E
4423 00        00800         DEFB     0
4424 54        00810 TM      DEFM     'TYPE MISMATCH'
4425 59
4426 50
4427 45
4428 20
4429 4D
442A 49
442B 53
442C 4D
442D 41
442E 54
442F 43
4430 48
4431 00        00820         DEFB     0
4432 4F        00830 OS      DEFM     'OUT OF STRING SPACE'
4433 55
4434 54
4435 20
4436 4F
4437 46
4438 20
```

```
4439 53
443A 54
443B 52
443C 49
443D 4E
443E 47
443F 20
4440 53
4441 50
4442 41
4443 43
4444 45
4445 00         00840        DEFB    0
4446 53         00850 LS     DEFM    'STRING TOO LONG'
4447 54
4448 52
4449 49
444A 4E
444B 47
444C 20
444D 54
444E 4F
444F 4F
4450 20
4451 4C
4452 4F
4453 4E
4454 47
4455 00         00860        DEFB    0
4456 53         00870 ST     DEFM    'STRING FORMULA TOO COMPLEX'
4457 54
4458 52
4459 49
445A 4E
445B 47
445C 20
445D 46
445E 4F
445F 52
4460 4D
4461 55
4462 4C
4463 41
4464 20
4465 54
4466 4F
4467 4F
4468 20
4469 43
446A 4F
446B 4D
446C 50
446D 4C
446E 45
446F 58
4470 00         00880        DEFB    0
4471 43         00890 CN     DEFM    'CAN'
4472 41
4473 4E
4474 27         00900        DEFB    39
4475 54         00910        DEFM    'T CONTINUE'
4476 20
4477 43
4478 4F
4479 4E
447A 54
447B 49
447C 4E
```

```
447D 55
447E 45
447F 00        00920        DEFB    0
4480 4E        00930 NR     DEFM    'NO RESUME'
4481 4F
4482 20
4483 52
4484 45
4485 53
4486 55
4487 4D
4488 45
4489 00        00940        DEFB    0
448A 52        00950 RW     DEFM    'RESUME WITHOUT ERROR'
448B 45
448C 53
448D 55
448E 4D
448F 45
4490 20
4491 57
4492 49
4493 54
4494 48
4495 4F
4496 55
4497 54
4498 20
4499 45
449A 52
449B 52
449C 4F
449D 52
449E 00        00960        DEFB    0
449F 55        00970 UE     DEFM    'UNPRINTABLE ERROR'
44A0 4E
44A1 50
44A2 52
44A3 49
44A4 4E
44A5 54
44A6 41
44A7 42
44A8 4C
44A9 45
44AA 20
44AB 45
44AC 52
44AD 52
44AE 4F
44AF 52
44B0 00        00980        DEFB    0
44B1 4D        00990 MO     DEFM    'MISSING OPERAND'
44B2 49
44B3 53
44B4 53
44B5 49
44B6 4E
44B7 47
44B8 20
44B9 4F
44BA 50
44BB 45
44BC 52
44BD 41
44BE 4E
44BF 44
44C0 00        01000        DEFB    0
```

```
44C1 42        01010 FD      DEFM    'BAD FILE DATA'
44C2 41
44C3 44
44C4 20
44C5 46
44C6 49
44C7 4C
44C8 45
44C9 20
44CA 44
44CB 41
44CC 54
44CD 41
44CE 00        01020         DEFB    0
44CF 44        01030 L3      DEFM    'DISK BASIC COMMAND'
44D0 49
44D1 53
44D2 4B
44D3 20
44D4 42
44D5 41
44D6 53
44D7 49
44D8 43
44D9 20
44DA 43
44DB 4F
44DC 4D
44DD 4D
44DE 41
44DF 4E
44E0 44
44E1 00        01040         DEFB    0
44E2 53        01050 M1      DEFM    'SPELLED OUT ERROR MESSAGES'
44E3 50
44E4 45
44E5 4C
44E6 4C
44E7 45
44E8 44
44E9 20
44EA 4F
44EB 55
44EC 54
44ED 20
44EE 45
44EF 52
44F0 52
44F1 4F
44F2 52
44F3 20
44F4 4D
44F5 45
44F6 53
44F7 53
44F8 41
44F9 47
44FA 45
44FB 53
44FC 0D        01060         DEFB    13
44FD 42        01070         DEFM    'BY MARK D. GOODWIN'
44FE 59
44FF 20
4500 4D
4501 41
4502 52
4503 4B
4504 20
```

```
4505 44
4506 2E
4507 20
4508 47
4509 4F
450A 4F
450B 44
450C 57
450D 49
450E 4E
450F 0D          01080          DEFB     13
4510 00          01090          DEFB     0
4511 00          01100 BASIC    DEFB     0
42E9             01110          END      START
00000 TOTAL ERRORS

L3       44CF
FD       44C1
MO       44B1
UE       449F
RW       448A
NR       4480
CN       4471
ST       4456
LS       4446
OS       4432
TM       4424
ID       440B
DO       43FA
DD       43E6
BS       43CF
UL       43B9
OM       43AB
OV       43A2
FC       438C
OD       4380
RG       436B
SN       435E
NF       434D
TABLE    431F
ERROR    4312
BASIC    4511
M1       44E2
START    42E9
```

Listing 9-1 Comments

100 - Set the program's starting address to the start of user RAM.
110 - Go clear the screen.
120 - Load register pair HL with the starting address of the message to be displayed.
130 - Go display the message.
140 - Load register pair HL with the start of the BASIC program area pointer.
150 - Zero the location of the start of the BASIC program area pointer in register pair HL.
160 - Bump the start of the BASIC program area pointer in register pair HL.
170 - Save the start of the BASIC program area pointer in register pair HL.

180 - Load register pair DE with the number of bytes to be reserved as string space.
190 - Go reserve the number of bytes of string space as specified by register pair DE.
200 - Go reset the BASIC variables and pointers.
210 - Load register A with a JP op code.
220 - Save the JP op code in register A as the first byte of the DOS link.
230 - Load register pair HL with the starting address of the spelled-out error message routine.
240 - Save the starting address of the spelled-out error message routine in register pair HL as the second and third bytes of the DOS link.
250 - Jump to the Level II BASIC READY routine.
260 - Zero register D, so that register pair DE will hold the error code.
270 - Load register pair HL with the starting address of the table of error message locations.
280 - Add the error code in register pair DE to the table pointer in register pair HL.
290 - Load register E with the LSB of the error message starting address at the location of the table pointer in register pair HL.
300 - Bump the table pointer in register pair HL.
310 - Load register D with the MSB of the error message starting address at the location of the table pointer in register pair HL.
320 - Load register pair HL with the starting address of the error message in register pair DE.
330 - Jump to the Level II error routine and display the appropriate error message.
340 - The starting address of the NF error message.
350 - The starting address of the SN error message.
360 - The starting address of the RG error message.
370 - The starting address of the OD error message.
380 - The starting address of the FC error message.
390 - The starting address of the OV error message.
400 - The starting address of the OM error message.
410 - The starting address of the UL error message.
420 - The starting address of the BC error message.
430 - The starting address of the DD error message.
440 - The starting address of the /O error message.

450 - The starting address of the ID error message.
460 - The starting address of the TM error message.
470 - The starting address of the OS error message.
480 - The starting address of the LS error message.
490 - The starting address of the ST error message.
500 - The starting address of the CN error message.
510 - The starting address of the NR error message.
520 - The starting address of the RW error message.
530 - The starting address of the UE error message.
540 - The starting address of the MO error message.
550 - The starting address of the FD error message.
560 - The starting address of the L3 error message.
570 - The NF error message is stored here.
580 - The NF error message terminator.
590 - The SN error message is stored here.
600 - The SN error message terminator.
610 - The RG error message is stored here.
620 - The RG error message terminator.
630 - The OD error message is stored here.
640 - The OD error message terminator.
650 - The FC error message is stored here.
660 - The FC error message terminator.
670 - The OV error message is stored here.
680 - The OV error message terminator.
690 - The OM error message is stored here.
700 - The OM error message terminator.
710 - The UL error message is stored here.
720 - The UL error message terminator.
730 - The BS error message is stored here.
740 - The BS error message terminator.
750 - The DD error message is stored here.
760 - The DD error message terminator.
770 - The /O error message is stored here.
780 - The /O error message terminator.
790 - The ID error message is stored here.
800 - The ID error message terminator.
810 - The TM error message is stored here.
820 - The TM error message terminator.
830 - The OS error message is stored here.
840 - The OS error message terminator.
850 - The LS error message is stored here.
860 - The LS error message terminator.

870 - The ST error message is stored here.
880 - The ST error message terminator.
890 - The CN error message starts here.
900 - This will display an ' with the CN error message.
910 - The rest of the CN error message is stored here.
920 - The CN error message terminator.
930 - The NR error message is stored here.
940 - The NR error message terminator.
950 - The RW error message is stored here.
960 - The RW error message terminator.
970 - The UE error message is stored here.
980 - The UE error message terminator.
990 - The MO error message is stored here.
1000 - The MO error message terminator.
1010 - The FD error message is stored here.
1020 - The FD error message terminator.
1030 - The L3 error message is stored here.
1040 - The L3 error message terminator.
1050 - Part of the program's sign-on message is stored here.
1060 - This will cause a carriage return to be displayed as part of the sign-on message.
1070 - Part of the sign-on message is stored here.
1080 - This will cause a carriage return to be displayed as part of the sign-on message.
1090 - The sign-on message terminator.
1100 - The start of the BASIC program area will begin here.
1110 - End of the program.

Chapter 10
A Machine Language Monitor

This chapter presents a relocatable machine language monitor. This monitor makes liberal use of the Level II BASIC ROM routines. Because of the program's size, it has been broken down into five parts, so that it can be assembled on a 16K computer.

THE MONITOR'S COMMANDS

The monitor has many useful commands. All of the monitor's commands require from 0 to 4 values to be input. The values should be entered as hexadecimal numbers. When more than one value is required for a command, the values should be separated by spaces. Leading zeros are not required. The syntax for each command follows the colon and does not include the period.

Display Memory: D starting address ending address. This command will display the contents of memory from the 'starting address' specified to the 'ending address' specified. The values, at these memory locations, will be displayed as two digit hexadecimal numbers. To end execution of this command before the ending address is reached, simply press the BREAK key. To pause the display, press the space bar. To resume execution, after pressing the space bar, press any key.

Jump to Address: G jump address. This command will cause program execution to begin at the 'jump address' specified.

Edit Memory: E starting address. This command will allow you to change the contents of memory. The command will begin by displaying the 'starting address' specified and the value currently at this memory address. To change the contents of an address, enter the desired value. To leave the contents of the address unchanged, press the carriage return without any input. To exit the Edit Memory mode, press the BREAK key.

Zero Memory: Z starting address ending address value. This command will change all of the memory locations from the 'starting address' specified to the 'ending address' specified to the 'value' given.

Input from Port: I port number. This command will get the value at the 'port' specified and display this value.

Output to Port: O port number value. This command will send the 'value' specified to the 'port' specified.

Convert Hexadecimal Numbers to Decimal Numbers: B first number second number. If only one number is input after the B command, this hexadecimal number will be converted to decimal and the result will be displayed. If two numbers are given, then they will be displayed like this.

A	B	A+B	A–B	B–A
7FFF	0001	8000	7FFE	8002
32767	1	32768	32766	32770

Set Breakpoint: @ breakpoint location. This command will set a breakpoint at the address specified. If no input follows the@command or an address of zero is given, then the current breakpoint location will be displayed.

Read a SYSTEM Tape: R. This command will load into memory a Level II BASIC SYSTEM tape. After the program has been completely loaded, the monitor will display the program's name, it's starting address, it's ending address, and it's execution address.

Write a SYSTEM Tape: P starting address ending address execution address. This command will prompt you to enter the file name for the program to be written on the cassette recorder. Once the file name has been entered, a Level II BASIC SYSTEM tape will be written beginning at the 'starting address' specified to the 'ending address' specified. If an 'execution address' is specified, then this address will also be written as part of the SYSTEM tape. If

an 'execution address' isn't specified, then the 'starting address' will be written as the program's execution address.

Display Characters in Memory: A starting address ending address. This command will display the ASCII characters for all of the locations from the 'starting address' specified to the 'ending address' specified.

Find 8-bit Value: F starting address ending address 8-bit value. This command will search the locations from the 'starting address' specified to the 'ending address' specified for the '8-bit value' given. All of the locations which contain the 8-bit value being searched for will be displayed.

Find the 16-bit Value: H starting address ending address 16-bit value. This command is the same as the F command except that it will search a block of memory for a 16-bit value.

Move Block of Memory: M starting address ending address new starting address. This command will move the block of memory from the 'starting address' specified to the 'ending address' specified to the new area of memory beginning at the 'new starting address.'

Figure Checksum: Q starting address ending address. This command will figure and display the checksum for the block of memory from the 'starting address' to the 'ending address.'

Test Memory: T starting address ending address. This command will test the block of memory from the 'starting address' to the 'ending address.' Any bad memory locations will be displayed.

Verify Memory: V starting address ending address second starting address. This command will compare the contents of a block of memory beginning at the 'starting address' to the 'ending address' to the second block of memory beginning at the 'second starting address.' Any memory locations which don't match will be displayed.

Exchange Memory: X starting address ending address second starting address. This command will exchange the contents of a block of memory beginning at the 'starting address' to the 'ending address' with the contents of a second block of memory beginning at the 'second starting address.'

Set the Current Output Device to Printer: L. This command will set the current output device to the printer.

Set the Current Output Device to Video Display: C. This command will set the current output device to the video display.

Save a Program on an ESF: "starting address ending address execution address file number. This command will save a program beginning at the 'starting address' to the 'ending address' on a Stringy Floppy wafer at the 'file number.' The 'execution address' will also be saved as the program's autostart address.

Certify an ESF Wafer: ! file number. This command will certify a Stringy Floppy wafer beginning at the 'file number.'

Load a Program from the ESF: # file number. This command will load a program from a Stringy Floppy wafer at the 'file number' given. The program's starting address, ending address, and execution address will be displayed after the program has been loaded.

THE MONITOR PROGRAM

The following pages contain the five parts of the monitor program (Listing 10-1 through 10-5). The line by line program comments follow each listing.

Listing 10-1

```
42F1              00100          ORG     42FFH-0EH
42F1  21B14A      00110 REND     LD      HL,4AB1H
42F4  ED5BB24A    00120          LD      DE,(4AB2H)
42F8  01B207      00130          LD      BC,1970
42FB  EDB8        00140          LDDR
42FD  EB          00150          EX      DE,HL
42FE  23          00160          INC     HL
42FF  E9          00170          JP      (HL)
4300              00180          END     4300H
00000 TOTAL ERRORS

REND      42F1
```

Listing 10-1 Comments

100 - Set the starting address of the program so that it resides below the monitor program.

110 - Load register pair HL with the end of the monitor program.

120 - Load register pair DE with the monitor program's new ending address.

130 - Load register pair BC with the length of the monitor program.

140 - Move the monitor program.

150 - Load register pair HL with the starting address of the monitor program – 1.

160 - Bump the starting address of the monitor in register pair HL.

170 - Jump to the location of the monitor's starting address in register pair HL.

180 - End of the program.

Listing 10-2

```
4300                      ORG     4300H
4300 21AB45       ST      LD      HL,M1
4303 CD752B               CALL    2B75H
4306 210000               LD      HL,0
4309 228E45               LD      (BREAK),HL
430C C32C46               JP      L1
430F AF           HEX     XOR     A
4310 328C45               LD      (FLAG),A
4313 110000               LD      DE,0
4316 23           H4      INC     HL
4317 7E                   LD      A,(HL)
4318 B7                   OR      A
4319 282A                 JR      Z,H1
431B D630                 SUB     30H
431D D8                   RET     C
431E FE0A                 CP      0AH
4320 3808                 JR      C,H3
4322 FE11                 CP      11H
4324 D8                   RET     C
4325 FE17                 CP      17H
4327 D0                   RET     NC
4328 D607                 SUB     7
432A F5           H3      PUSH    AF
432B 3E01                 LD      A,1
432D 328C45               LD      (FLAG),A
4330 F1                   POP     AF
4331 CB27                 SLA     A
4333 CB27                 SLA     A
4335 CB27                 SLA     A
4337 CB27                 SLA     A
4339 0604                 LD      B,4
433B CB27         H5      SLA     A
433D CB13                 RL      E
433F CB12                 RL      D
4341 10F8                 DJNZ    H5
4343 18D1                 JR      H4
4345 2B           H1      DEC     HL
4346 C9                   RET
4347 CD9F44       DISP    CALL    CH1
434A 2A8545               LD      HL,(N1)
434D CD8544       DI1     CALL    DIS1
4350 0610                 LD      B,16
4352 7E           DI2     LD      A,(HL)
4353 CD9244               CALL    DSP
4356 CD2046               CALL    CH2
4359 10F7                 DJNZ    DI2
435B CD8044               CALL    CR
```

```
435E CD1146   00570          CALL    KEY
4361 18EA     00580          JR      DI1
4363 E5       00590 D1       PUSH    HL
4364 328B45   00600          LD      (N4),A
4367 AF       00610          XOR     A
4368 218B45   00620          LD      HL,N4
436B ED6F     00630          RLD
436D CD7843   00640          CALL    D2
4370 AF       00650          XOR     A
4371 ED6F     00660          RLD
4373 CD7843   00670          CALL    D2
4376 E1       00680          POP     HL
4377 C9       00690          RET
4378 C630     00700 D2       ADD     A,30H
437A FE3A     00710          CP      3AH
437C 3802     00720          JR      C,D3
437E C607     00730          ADD     A,7
4380 C32A03   00740 D3       JP      32AH
4383 2A8545   00750 GO       LD      HL,(N1)
4386 E9       00760          JP      (HL)
4387 2A8545   00770 EDIT     LD      HL,(N1)
438A CD8544   00780 E1       CALL    DIS1
438D 7E       00790          LD      A,(HL)
438E 328545   00800          LD      (N1),A
4391 CD9244   00810          CALL    DSP
4394 E5       00820          PUSH    HL
4395 CD6103   00830          CALL    361H
4398 DA2C46   00840          JP      C,L1
439B D7       00850          RST     16
439C 2B       00860          DEC     HL
439D CD0F43   00870          CALL    HEX
43A0 3A8C45   00880          LD      A,(FLAG)
43A3 B7       00890          OR      A
43A4 2804     00900          JR      Z,E3
43A6 ED538545 00910          LD      (N1),DE
43AA 3A8545   00920 E3       LD      A,(N1)
43AD E1       00930          POP     HL
43AE 77       00940          LD      (HL),A
43AF 23       00950          INC     HL
43B0 18D8     00960          JR      E1
43B2 2A8545   00970 ZERO     LD      HL,(N1)
43B5 3A8945   00980 Z1       LD      A,(N3)
43B8 77       00990          LD      (HL),A
43B9 ED5B8745 01000          LD      DE,(N2)
43BD DF       01010          RST     18H
43BE CA2C46   01020          JP      Z,L1
43C1 23       01030          INC     HL
43C2 18F1     01040          JR      Z1
43C4 CDA244   01050 INP      CALL    PR
43C7 3A8545   01060          LD      A,(N1)
43CA 4F       01070          LD      C,A
43CB ED78     01080          IN      A,(C)
43CD CD6343   01090          CALL    D1
43D0 CD2946   01100          CALL    CH3
43D3 3A8545   01110 OUP      LD      A,(N1)
43D6 4F       01120          LD      C,A
43D7 3A8745   01130          LD      A,(N2)
43DA ED79     01140          OUT     (C),A
43DC C32C46   01150          JP      L1
43DF CDA244   01160 MATH     CALL    PR
43E2 CDA944   01170 MA3      CALL    LO
43E5 7A       01180          LD      A,D
43E6 B3       01190          OR      E
43E7 CA4E44   01200          JP      Z,MA1
43EA 21CB45   01210          LD      HL,M2
43ED CD752B   01220          CALL    2B75H
43F0 2A8545   01230          LD      HL,(N1)
43F3 CD7644   01240          CALL    DIWS
```

```
43F6 2A8745     01250          LD      HL,(N2)
43F9 CD7644     01260          CALL    DIWS
43FC CDA944     01270          CALL    LO
43FF 19         01280          ADD     HL,DE
4400 229345     01290          LD      (REG),HL
4403 CD7644     01300          CALL    DIWS
4406 CDA944     01310          CALL    LO
4409 AF         01320          XOR     A
440A ED52       01330          SBC     HL,DE
440C 229545     01340          LD      (REG+2),HL
440F CD7644     01350          CALL    DIWS
4412 CDA944     01360          CALL    LO
4415 EB         01370          EX      DE,HL
4416 AF         01380          XOR     A
4417 ED52       01390          SBC     HL,DE
4419 229745     01400          LD      (REG+4),HL
441C CD7644     01410          CALL    DIWS
441F CD8044     01420          CALL    CR
4422 2A8545     01430          LD      HL,(N1)
4425 1E06       01440          LD      E,6
4427 CD5944     01450          CALL    TAB
442A 2A8745     01460          LD      HL,(N2)
442D 1E0C       01470          LD      E,12
442F CD5944     01480          CALL    TAB
4432 2A9345     01490          LD      HL,(REG)
4435 1E12       01500          LD      E,18
4437 CD5944     01510          CALL    TAB
443A 2A9545     01520          LD      HL,(REG+2)
443D 1E18       01530          LD      E,24
443F CD5944     01540          CALL    TAB
4442 2A9745     01550          LD      HL,(REG+4)
4445 CDAF0F     01560          CALL    0FAFH
4448 CD8044     01570          CALL    CR
444B C32C46     01580 MA2      JP      L1
444E 2A8545     01590 MA1      LD      HL,(N1)
4451 CDAF0F     01600          CALL    0FAFH
4454 CD8044     01610          CALL    CR
4457 18F2       01620          JR      MA2
4459 D5         01630 TAB      PUSH    DE
445A CDAF0F     01640          CALL    0FAFH
445D D1         01650          POP     DE
445E 3A9C40     01660          LD      A,(409CH)
4461 B7         01670          OR      A
4462 2805       01680          JR      Z,VID
4464 3A9B40     01690          LD      A,(409BH)
4467 1803       01700          JR      T1
4469 3AA640     01710 VID      LD      A,(40A6H)
446C 2F         01720 T1       CPL
446D 83         01730          ADD     A,E
446E 3C         01740          INC     A
446F 47         01750          LD      B,A
4470 CD8D44     01760 T2       CALL    SPA
4473 10FB       01770          DJNZ    T2
4475 C9         01780          RET
4476 CD7B44     01790 DIWS     CALL    DIS
4479 1812       01800          JR      SPA
447B CD9744     01810 DIS      CALL    DISW
447E 180D       01820          JR      SPA
4480 3E0D       01830 CR       LD      A,13
4482 C32A03     01840          JP      32AH
4485 CD9744     01850 DIS1     CALL    DISW
4488 3E3A       01860          LD      A,':'
448A CD2A03     01870          CALL    32AH
448D 3E20       01880 SPA      LD      A,' '
448F C32A03     01890          JP      32AH
4492 CD6343     01900 DSP      CALL    D1
4495 18F6       01910          JR      SPA
4497 7C         01920 DISW     LD      A,H
```

```
4498 CD6343               CALL   D1
449B 7D                   LD     A,L
449C C36343               JP     D1
449F CDD144     CH1       CALL   CHECK
44A2 3A8D45     PR        LD     A,(FL2)
44A5 329C40               LD     (409CH),A
44A8 C9                   RET
44A9 2A8545     LO        LD     HL,(N1)
44AC ED5B8745             LD     DE,(N2)
44B0 C9                   RET
44B1 0604       CH4       LD     B,4
44B3 CDC544     CHA4      CALL   CH5
44B6 10FB                 DJNZ   CHA4
44B8 C9                   RET
44B9 CDC544     CH6       CALL   CH5
44BC DD23                 INC    IX
44BE DD23                 INC    IX
44C0 CDC544               CALL   CH5
44C3 18BB                 JR     CR
44C5 DD6E00     CH5       LD     L,(IX+0)
44C8 DD23                 INC    IX
44CA DD6600               LD     H,(IX+0)
44CD DD23                 INC    IX
44CF 18AA                 JR     DIS
44D1 2A8545     CHECK     LD     HL,(N1)
44D4 7C                   LD     A,H
44D5 B5                   OR     L
44D6 C0                   RET    NZ
44D7 2A8745               LD     HL,(N2)
44DA 7C                   LD     A,H
44DB B5                   OR     L
44DC C0                   RET    NZ
44DD 21FFFF               LD     HL,0FFFFH
44E0 228745               LD     (N2),HL
44E3 C9                   RET
44E4 2A8545     BRK       LD     HL,(N1)
44E7 7C                   LD     A,H
44E8 B5                   OR     L
44E9 2819                 JR     Z,B1
44EB E5                   PUSH   HL
44EC 228E45               LD     (BREAK),HL
44EF 119045               LD     DE,SAVE
44F2 0603                 LD     B,3
44F4 7E         B2        LD     A,(HL)
44F5 12                   LD     (DE),A
44F6 23                   INC    HL
44F7 13                   INC    DE
44F8 10FA                 DJNZ   B2
44FA E1                   POP    HL
44FB 36CD                 LD     (HL),0CDH
44FD 23                   INC    HL
44FE 111645               LD     DE,RENT
4501 73                   LD     (HL),E
4502 23                   INC    HL
4503 72                   LD     (HL),D
4504 21E845     B1        LD     HL,RE1
4507 CD752B               CALL   2B75H
450A 2A8E45               LD     HL,(BREAK)
450D CD9744               CALL   DISW
4510 CD8044               CALL   CR
4513 C32C46               JP     L1
4516 22A745     RENT      LD     (REG+20),HL
4519 E1                   POP    HL
451A 2B                   DEC    HL
451B 2B                   DEC    HL
451C 2B                   DEC    HL
451D 22A945               LD     (REG+22),HL
```

```
4520 2AA745   02600          LD      HL,(REG+20)
4523 ED73A745 02610          LD      (REG+20),SP
4527 31A745   02620          LD      SP,REG+20
452A FDE5     02630          PUSH    IY
452C DDE5     02640          PUSH    IX
452E E5       02650          PUSH    HL
452F D5       02660          PUSH    DE
4530 C5       02670          PUSH    BC
4531 F5       02680          PUSH    AF
4532 08       02690          EX      AF,AF'
4533 D9       02700          EXX
4534 E5       02710          PUSH    HL
4535 D5       02720          PUSH    DE
4536 C5       02730          PUSH    BC
4537 F5       02740          PUSH    AF
4538 310043   02750          LD      SP,ST
453B CDA244   02760          CALL    PR
453E 2A8E45   02770          LD      HL,(BREAK)
4541 119045   02780          LD      DE,SAVE
4544 0603     02790          LD      B,3
4546 1A       02800 R1       LD      A,(DE)
4547 77       02810          LD      (HL),A
4548 23       02820          INC     HL
4549 13       02830          INC     DE
454A 10FA     02840          DJNZ    R1
454C 21E845   02850          LD      HL,RE1
454F CD752B   02860          CALL    2B75H
4552 2A8E45   02870          LD      HL,(BREAK)
4555 CD9744   02880          CALL    DISW
4558 CD8044   02890          CALL    CR
455B CD8044   02900          CALL    CR
455E 21F245   02910          LD      HL,RE2
4561 CD752B   02920          CALL    2B75H
4564 DD219B45 02930          LD      IX,REG+8
4568 CDB144   02940          CALL    CH4
456B CDB944   02950          CALL    CH6
456E DD219345 02960          LD      IX,REG
4572 CDB144   02970          CALL    CH4
4575 DD21A545 02980          LD      IX,REG+18
4579 CDB944   02990          CALL    CH6
457C 210000   03000          LD      HL,0
457F 228E45   03010          LD      (BREAK),HL
4582 C32C46   03020          JP      L1
4585 0000     03030 N1       DEFW    0
4587 0000     03040 N2       DEFW    0
4589 0000     03050 N3       DEFW    0
458B 00       03060 N4       DEFB    0
458C 00       03070 FLAG     DEFB    0
458D 00       03080 FL2      DEFB    0
458E 0000     03090 BREAK    DEFW    0
0003          03100 SAVE     DEFS    3
0018          03110 REG      DEFS    24
45AB 1C       03120 M1       DEFB    1CH
45AC 1F       03130          DEFB    1FH
45AD 53       03140          DEFM    'SWAT V2.0'
45AE 57
45AF 41
45B0 54
45B1 20
45B2 56
45B3 32
45B4 2E
45B5 30
45B6 0D       03150          DEFB    13
45B7 42       03160          DEFM    'BY MARK D. GOODWIN'
45B8 59
45B9 20
45BA 4D
```

```
45BB 41
45BC 52
45BD 4B
45BE 20
45BF 44
45C0 2E
45C1 20
45C2 47
45C3 4F
45C4 4F
45C5 44
45C6 57
45C7 49
45C8 4E
45C9 0D      03170          DEFB    13
45CA 00      03180          NOP
45CB 41      03190 M2       DEFM    'A     B     A+B   A-B   B-A'
45CC 20
45CD 20
45CE 20
45CF 20
45D0 20
45D1 42
45D2 20
45D3 20
45D4 20
45D5 20
45D6 20
45D7 41
45D8 2B
45D9 42
45DA 20
45DB 20
45DC 20
45DD 41
45DE 2D
45DF 42
45E0 20
45E1 20
45E2 20
45E3 42
45E4 2D
45E5 41
45E6 0D      03200          DEFB    13
45E7 00      03210          NOP
45E8 42      03220 RE1      DEFM    'BREAK AT '
45E9 52
45EA 45
45EB 41
45EC 4B
45ED 20
45EE 41
45EF 54
45F0 20
45F1 00      03230          NOP
45F2 41      03240 RE2      DEFM    'AF   BC   DE   HL
                            IX/Y STAK'
45F3 46
45F4 20
45F5 20
45F6 20
45F7 42
45F8 43
45F9 20
45FA 20
45FB 20
45FC 44
45FD 45
```

```
45FE 20
45FF 20
4600 20
4601 48
4602 4C
4603 20
4604 20
4605 20
4606 49
4607 58
4608 2F
4609 59
460A 20
460B 53
460C 54
460D 41
460E 4B
460F 0D         03250          DEFB     13
4610 00         03260          NOP
4611 CD2B00     03270 KEY      CALL     2BH
4614 B7         03280          OR       A
4615 C8         03290          RET      Z
4616 FE01       03300          CP       1
4618 2812       03310          JR       Z,L1
461A FE20       03320          CP       ' '
461C C0         03330          RET      NZ
461D C34900     03340          JP       49H
4620 ED5B8745   03350 CH2      LD       DE,(N2)
4624 DF         03360          RST      18H
4625 2802       03370          JR       Z,CH3
4627 23         03380          INC      HL
4628 C9         03390          RET
4629 CD8044     03400 CH3      CALL     CR
462C 00         03410 L1       DEFB     0
4300            03420          END      ST
00000 TOTAL ERRORS

RE2      45F2
R1       4546
RE1      45E8
RENT     4516
B2       44F4
SAVE     4590
B1       4504
BRK      44E4
CH6      44B9
CH5      44C5
CHA4     44B3
CH4      44B1
FL2      458D
CHECK    44D1
DISW     4497
DIS      447B
SPA      448D
T2       4470
T1       446C
VID      4469
MA2      444B
TAB      4459
REG      4593
DIWS     4476
M2       45CB
MA1      444E
LO       44A9
MA3      43E2
MATH     43DF
OUP      43D3
CH3      4629
```

```
PR      44A2
INP     43C4
N2      4587
N3      4589
Z1      43B5
ZERO    43B2
E3      43AA
E1      438A
EDIT    4387
GO      4383
D3      4380
D2      4378
N4      458B
D1      4363
KEY     4611
CR      4480
CH2     4620
DSP     4492
DI2     4352
DIS1    4485
DI1     434D
N1      4585
CH1     449F
DISP    4347
H5      433B
H3      432A
H1      4345
H4      4316
FLAG    458C
HEX     430F
L1      462C
BREAK   458E
M1      45AB
ST      4300
```

Listing 10-2 Comments

100 - Set the starting address for the monitor program.
110 - Load register pair HL with the starting address of the message to be displayed.
120 - Go display the message.
130 - Zero register pair HL.
140 - Save the value in register pair HL at the breakpoint address location.
150 - Jump.
160 - Zero register A.
170 - Zero the input flag.
180 - Zero register pair DE.
190 - Bump the input pointer in register pair HL.
200 - Load register A with the character at the location of the input pointer in register pair HL.
210 - Check to see if this is the end of the input.
220 - Jump if this is the end of the input.
230 - Check to see if the character in register A is less than a 0.
240 - Return if the character in register A is less than a 0.
250 - Check to see if the character in register A is numeric.

260 - Jump if the character in register A is numeric.
270 - Check to see if the character in register A is less than an A.
280 - Return if the character in register A is less than an A.
290 - Check to see if the character in register A is greater than a F.
300 - Return if the character in register A is greater than a F.
310 - Adjust the letter in register A so that it will be a binary 10-15.
320 - Save the binary value in register A on the stack.
330 - Load register A with a 1.
340 - Save the value in register A as the current input flag.
350 - Get the binary value from the stack and put it in register A.
360 - Move the binary value in register A over one bit.
370 - Move the binary value in register A over one bit.
380 - Move the binary value in register A over one bit.
390 - Move the binary value in register A over one bit.
400 - Load register B with the number of bits to be moved into register pair DE.
410 - Move the most significant bit in register A into the Carry flag.
420 - Move the bit in the Carry flag into register E and move register E's most significant bit into the Carry flag.
430 - Move the bit in the Carry flag into register D.
440 - Loop till all four bits of the hex digit have been moved into register pair DE.
450 - Loop till the end of the hex number has been processed.
460 - Decrement the input pointer in register pair HL.
470 - Return with the binary value of the hex number in register pair DE.
480 - Go check the addresses and set the current output device.
490 - Load register pair HL with the starting address to be displayed.
500 - Go send the value in register pair HL to the current output device.
510 - Load register B with the number of memory locations per line to output.
520 - Load register A with the value at the location of the memory pointer in register pair HL.
530 - Go display the value in register A as two hex digits.
540 - Check to see if the memory pointer in register pair HL is the same as the last address.
550 - Loop till 16 values have been sent to the current output device.

560 - Go display a carriage return.
570 - Go check to see if the BREAK or the space key was pressed.
580 - Loop till done.
590 - Save the value in register pair HL on the stack.
600 - Save the value in register A.
610 - Zero register A.
620 - Point register pair HL to the storage location of the value to be displayed.
630 - Load register A with the most significant four bits at the location in register pair HL.
640 - Go display the 4-bit value in register A.
650 - Zero register A.
660 - Load register A with the most significant four bits at the location in register pair HL.
670 - Go display the 4-bit value in register A.
680 - Get the value from the stack and put it in register pair HL.
690 - Return.
700 - Adjust the 4-bit value in register A so that it will be an ASCII character.
710 - Check to see if the character in register A is a 0-9.
720 - Jump if the character in register A is a 0-9.
730 - Adjust the 4-bit value in register A so that it will be the proper ASCII value for A-F.
740 - Go send the character in register A to the current output device.
750 - Load register pair HL with the location to jump to.
760 - Jump to the location in register pair HL.
770 - Load register pair HL with the starting address.
780 - Go display the address in register pair HL.
790 - Load register A with the value at the location of the memory pointer in register pair HL.
800 - Save the value in register A.
810 - Go display the value in register A.
820 - Save the memory pointer in register pair HL on the stack.
830 - Go get the input from the keyboard.
840 - Jump if BREAK is pressed.
850 - Point the input pointer in register pair HL to the first character in the input.
860 - Decrement the input pointer in register pair HL.
870 - Go convert the hex input to binary.

880 - Load register A with the input flag.
890 - Check to see if a hex number was input.
900 - Jump if a hex number wasn't input.
910 - Save the binary value for the hex input in register pair DE.
920 - Load register A with the value to be saved.
930 - Get the memory pointer from the stack and put it in register pair HL.
940 - Save the value in register A at the location of the memory pointer in register pair HL.
950 - Bump the memory pointer in register pair HL.
960 - Loop till the BREAK key is pressed.
970 - Load register pair HL with the starting address.
980 - Load register A with the value to save.
990 - Save the value in register A at the location of the memory pointer in register pair HL.
1000 - Load register pair DE with the ending address.
1010 - Go compare the memory pointer in register pair HL with the ending address in register pair DE.
1020 - Jump if the memory pointer in register pair HL is the same as the ending address in register pair DE.
1030 - Bump the memory pointer in register pair HL.
1040 - Loop till done.
1050 - Go set the current output device.
1060 - Load register A with the port to input from.
1070 - Load register C with the port number in register A.
1080 - Go read the port.
1090 - Go display the value read in register A.
1100 - Go send a carriage return to the current output device and return.
1110 - Load register A with the port number.
1120 - Load register C with the port number in register A.
1130 - Load register A with the value to send to the port.
1140 - Send the value in register A to the port in register C.
1150 - Jump.
1160 - Go set the current output device.
1170 - Go load the first value input into register pair HL and the second value input into register pair DE.
1180 - Load register A with the MSB of the second value input in register D.
1190 - Combine the MSB of the second value input in register A with the LSB of the second value input in register E.

1200 - Jump if there wasn't a second value input.

1210 - Load register pair HL with the starting address of the message to be displayed.

1220 - Go display the message.

1230 - Load register pair HL with the first number.

1240 - Go display the number in register pair HL.

1250 - Load register pair HL with the second number.

1260 - Go display the second number in register pair HL.

1270 - Go load the first number in register pair HL and the second number in register pair DE.

1280 - Add the second number in register pair DE to the first number in register pair HL.

1290 - Save the result in register pair HL.

1300 - Go display the result in register pair HL.

1310 - Load the first number in register pair HL and the second number in register pair DE.

1320 - Clear the Carry flag.

1330 - Go subtract the second number in register pair DE from the first number in register pair HL.

1340 - Save the result in register pair HL.

1350 - Go display the result in register pair HL.

1360 - Go load the first number in register pair HL and the second number in register pair DE.

1370 - Exchange the first number in register pair HL with the second number in register pair DE.

1380 - Clear the Carry flag.

1390 - Subtract the first number in register pair DE from the second number in register pair HL.

1400 - Save the result in register pair HL.

1410 - Go display the result in register pair HL.

1420 - Go display a carriage return.

1430 - Load register pair HL with the first number.

1440 - Load register E with the tab position after displaying the result in register pair HL.

1450 - Go display the number in register pair HL and tab to the next position.

1460 - Load register pair HL with the second number.

1470 - Load register E with the next tab position after displaying the number in register pair HL.

1480 - Go display the number in register pair HL and tab to the next position.

1490 - Load register pair HL with the result of adding the first and second numbers.
1500 - Load register E with the next tab position after displaying the result in register pair HL.
1510 - Go display the result in register pair HL and tab to the next position.
1520 - Load register pair HL with the result of subtracting the second number from the first number.
1530 - Load register E with the next tab position after displaying the result in register pair HL.
1540 - Go display the result in register pair HL and tab to the next position.
1550 - Load register pair HL with the result of subtracting the first number from the second number.
1560 - Go display the result in register pair HL.
1570 - Go display a carriage return.
1580 - Jump.
1590 - Load register pair HL with the number to be converted.
1600 - Go display the number in register pair HL.
1610 - Go display a carriage return.
1620 - Jump.
1630 - Save the tab position in register E on the stack.
1640 - Go display the number in register pair HL.
1650 - Get the tab position from the stack and put it in register E.
1660 - Load register A with the current output device.
1670 - Check to see if the current output device is the video display.
1680 - Jump if the current output device is the video display.
1690 - Load register A with the current carriage position.
1700 - Jump.
1710 - Load register A with the current cursor position.
1720 - Compliment the current print position in register A.
1730 - Figure the number of spaces to be printed.
1740 - Bump the number of spaces to be printed in register A.
1750 - Load register B with the number of spaces to be printed in register A.
1760 - Go print a space.
1770 - Loop till all of the spaces have been printed.
1780 - Return.
1790 - Go display the value in register pair HL and a space.
1800 - Go display a space.

1810 - Go print the value in register pair HL.
1820 - Go display a space.
1830 - Load register A with a carriage return character.
1840 - Go display a carriage return on the current output device.
1850 - Go display the value in register pair HL.
1860 - Load register A with a : character.
1870 - Go send the : character in register A to the current output device.
1880 - Load register A with a space character.
1890 - Go send the space in register A to the current output device.
1900 - Go display the 8-bit value in register A.
1910 - Go display a space.
1920 - Load register A with the MSB of the value to be displayed in register H.
1930 - Go display the 8-bit value in register A.
1940 - Load register A with the LSB of the value to be displayed in register L.
1950 - Go display the 8-bit value in register A.
1960 - Go check the numbers that were input and set the default values if necessary.
1970 - Load register A with the output device number.
1980 - Save the value in register A as the current output device flag.
1990 - Return.
2000 - Load register pair HL with the first number.
2010 - Load register pair DE with the second number.
2020 - Return.
2030 - Load register B with the number of values to be displayed.
2040 - Go display the value at the location of the memory pointer in register pair IX.
2050 - Loop till all of the numbers have been displayed.
2060 - Return.
2070 - Go display the value at the location of the memory pointer in register pair IX.
2080 - Bump the memory pointer in register pair IX.
2090 - Bump the memory pointer in register pair IX.
2100 - Go display the value at the location of the memory pointer in register pair IX.
2110 - Go display a carriage return.
2120 - Load register L with the LSB of the value to be displayed at

the location of the memory pointer in register pair IX.
2130 - Bump the memory pointer in register pair IX.
2140 - Load register H with the MSB of the value to be displayed at the location of the memory pointer in register pair IX.
2150 - Bump the memory pointer in register pair IX.
2160 - Go display the value in register pair HL.
2170 - Load register pair HL with the first number input.
2180 - Load register A with the MSB of the first number input in register H.
2190 - Combine the MSB of the first number input in register A with the LSB of the first number input in register L.
2200 - Return if the first number input isn't equal to zero.
2210 - Load register pair HL with the second number input.
2220 - Load register A with the MSB of the second number input in register H.
2230 - Combine the MSB of the second number input in register A with the LSB of the second number input in register L.
2240 - Return if the second number input isn't equal to zero.
2250 - Load register pair HL with the end of memory.
2260 - Save the value in register pair HL as the second number input.
2270 - Return.
2280 - Load register pair HL with the first number input.
2290 - Load register A with the MSB of the first number input in register H.
2300 - Combine the MSB of the first number input in register A with the LSB of the first number input in register L.
2310 - Jump if the first number input is equal to zero.
2320 - Save the first number input in register pair HL on the stack.
2330 - Save the first number input in register pair HL as the current breakpoint location.
2340 - Load register pair DE with the location to save the values at the breakpoint location.
2350 - Load register B with the number of values to be saved.
2360 - Load register A with the value at the breakpoint pointer in register pair HL.
2370 - Save the value in register A at the location of the breakpoint storage pointer in register pair DE.
2380 - Bump the breakpoint pointer in register pair HL.
2390 - Bump the breakpoint storage pointer in register pair DE.
2400 - Loop till the three bytes at the breakpoint location have been saved.

2410 - Get the location of the breakpoint from the stack and put it in register pair HL.
2420 - Save a call op code at the location of the breakpoint pointer in register pair HL.
2430 - Bump the breakpoint pointer in register pair HL.
2440 - Load register pair DE with the reentry point.
2450 - Save the LSB of the reentry point in register E at the location of the breakpoint pointer in register pair HL.
2460 - Bump the breakpoint pointer in register pair HL.
2470 - Save the MSB of the reentry point in register D at the location of the breakpoint pointer in register pair HL.
2480 - Load register pair HL with the starting address of the message to be displayed.
2490 - Go display the message.
2500 - Load register pair HL with the current breakpoint location.
2510 - Go display the current breakpoint location in register pair HL.
2520 - Go display a carriage return.
2530 - Jump.
2540 - Save the value in register pair HL.
2550 - Get the return address from the stack and put it in register pair HL.
2560 - Decrement the return address in register pair HL.
2570 - Decrement the return address in register pair HL.
2580 - Decrement the return address in register pair HL.
2590 - Save the return address in register pair HL.
2600 - Load register pair HL with the value previously saved.
2610 - Save the current location of the stack pointer.
2620 - Set the stack pointer at the storage locations for the registers.
2630 - Save the value in register pair IY.
2640 - Save the value in register pair IX.
2650 - Save the value in register pair HL.
2660 - Save the value in register pair DE.
2670 - Save the value in register pair BC.
2680 - Save the value in register pair AF.
2690 - Exchange the prime and the nonprime registers.
2700 - Exchange the prime and the nonprime registers.
2710 - Save the value in register pair HL.
2720 - Save the value in register pair DE.
2730 - Save the value in register pair BC.
2740 - Save the value in register pair AF.

2750 - Reset the stack pointer.
2760 - Go set the current output device.
2770 - Load register pair HL with the current location of the breakpoint.
2780 - Load register pair DE with the location of the breakpoint storage location.
2790 - Load register B with the number of bytes to be replaced in memory.
2800 - Load register A with the value at the location of the breakpoint storage location in register pair DE.
2810 - Save the value in register A at the location of the breakpoint pointer in register pair HL.
2820 - Bump the breakpoint pointer in register pair HL.
2830 - Bump the breakpoint storage pointer in register pair DE.
2840 - Loop till all of the bytes have been replaced.
2850 - Load register pair HL with the starting address of the message to be displayed.
2860 - Go display the message.
2870 - Load register pair HL with the current breakpoint location.
2880 - Go display the current breakpoint location in register pair HL.
2890 - Go display a carriage return.
2900 - Go display a carriage return.
2910 - Load register pair HL with the starting address of the message to be displayed.
2920 - Go display the message.
2930 - Load register pair IX with the register storage location.
2940 - Go display the register values.
2950 - Go display the register values.
2960 - Load register pair IX with the location of the registers to be displayed.
2970 - Go display the register values.
2980 - Load register pair IX with the register values storage location.
2990 - Go display the register values.
3000 - Zero register pair HL.
3010 - Save the value in register pair HL as the current breakpoint location.
3020 - Jump.
3030 - The first number input is stored here.
3040 - The second number input is stored here.
3050 - The third number input is stored here.

3060 - Temporary storage location.
3070 - Input flag.
3080 - Current output device number is stored here.
3090 - The current breakpoint location is stored here.
3100 - The values at the current breakpoint location are stored here.
3110 - The register values are stored here upon reentry from a breakpoint.
3120 - This will home the cursor.
3130 - This will clear to the end of the screen.
3140 - Part of the sign-on message is stored here.
3150 - This will display a carriage return as part of the sign-on message.
3160 - Part of the sign-on message is stored here.
3170 - This will display a carriage return as part of the sign-on message.
3180 - Sign-on message terminator.
3190 - Math conversion message is stored here.
3200 - This will display a carriage return as part of the math conversion message.
3210 - Math conversion message terminator is stored here.
3220 - Breakpoint message is stored here.
3230 - Breakpoint message terminator is stored here.
3240 - Register display message is stored here.
3250 - This will display a carriage return as part of the register display message.
3260 - Register display message terminator is stored here.
3270 - Go scan the keyboard.
3280 - Check to see if there was a key pressed.
3290 - Return if a key wasn't pressed.
3300 - Check to see if the BREAK key was pressed.
3310 - Jump if the BREAK key was pressed.
3320 - Check to see if the space key was pressed.
3330 - Return if the space key wasn't pressed.
3340 - Loop till another key is pressed.
3350 - Load register pair DE with the second number input.
3360 - Go check to see if the current value in register pair HL is the same as the second number input in register pair DE.
3370 - Jump if the value in register pair HL is the same as the second number input in register pair DE.
3380 - Bump the value in register pair HL.
3390 - Return.

3400 - Go display a carriage return.
3410 - Link to the next part.
3420 - End of the program.

Listing 10-3

```
4585           00100 N1     EQU    4585H
4587           00110 N2     EQU    4587H
4589           00120 N3     EQU    4589H
458C           00130 FLAG   EQU    458CH
4497           00140 DISW   EQU    4497H
4480           00150 CR     EQU    4480H
44D1           00160 CHECK  EQU    44D1H
44A2           00170 PR     EQU    44A2H
4485           00180 DIS1   EQU    4485H
4492           00190 DSP    EQU    4492H
4363           00200 D1     EQU    4363H
448D           00210 SPA    EQU    448DH
449F           00220 CH1    EQU    449FH
4620           00230 CH2    EQU    4620H
4629           00240 CH3    EQU    4629H
4611           00250 KEY    EQU    4611H
462C           00260        ORG    462CH
462C C3E948    00270        JP     L1
462F AF        00280 READ   XOR    A
4630 328C45    00290        LD     (FLAG),A
4633 CDBC46    00300        CALL   CASS
4636 CD9602    00310        CALL   296H
4639 CD3502    00320 RA1    CALL   235H
463C FE55      00330        CP     55H
463E 20F9      00340        JR     NZ,RA1
4640 0606      00350        LD     B,6
4642 21D246    00360        LD     HL,NAME
4645 CD3502    00370 RA2    CALL   235H
4648 77        00380        LD     (HL),A
4649 23        00390        INC    HL
464A 10F9      00400        DJNZ   RA2
464C CD2C02    00410 RA3    CALL   22CH
464F CD3502    00420 RA4    CALL   235H
4652 FE78      00430        CP     78H
4654 2835      00440        JR     Z,RA6
4656 FE3C      00450        CP     3CH
4658 20F5      00460        JR     NZ,RA4
465A CD3502    00470        CALL   235H
465D 47        00480        LD     B,A
465E CD1403    00490        CALL   314H
4661 85        00500        ADD    A,L
4662 4F        00510        LD     C,A
4663 3A8C45    00520        LD     A,(FLAG)
4666 FE01      00530        CP     1
4668 2808      00540        JR     Z,RA5
466A 228545    00550        LD     (N1),HL
466D 3E01      00560        LD     A,1
466F 328C45    00570        LD     (FLAG),A
4672 CD3502    00580 RA5    CALL   235H
4675 77        00590        LD     (HL),A
4676 23        00600        INC    HL
4677 228745    00610        LD     (N2),HL
467A 81        00620        ADD    A,C
467B 4F        00630        LD     C,A
467C 10F4      00640        DJNZ   RA5
467E CD3502    00650        CALL   235H
4681 B9        00660        CP     C
4682 28C8      00670        JR     Z,RA3
```

```
4684 3E43     00680          LD     A,43H
4686 323E3C   00690          LD     (3C3EH),A
4689 18C4     00700          JR     RA4
468B CD1403   00710 RA6      CALL   314H
468E 228945   00720          LD     (N3),HL
4691 CDF801   00730          CALL   1F8H
4694 21E946   00740          LD     HL,CA2
4697 CD752B   00750          CALL   2B75H
469A 21D246   00760          LD     HL,NAME
469D CD752B   00770          CALL   2B75H
46A0 CD8044   00780          CALL   CR
46A3 2A8545   00790          LD     HL,(N1)
46A6 CDCA46   00800          CALL   DIS2
46A9 2A8745   00810          LD     HL,(N2)
46AC 2B       00820          DEC    HL
46AD CDCA46   00830          CALL   DIS2
46B0 2A8945   00840          LD     HL,(N3)
46B3 CD9744   00850          CALL   DISW
46B6 CD8044   00860          CALL   CR
46B9 C3E948   00870          JP     L1
46BC 21D946   00880 CASS     LD     HL,CA1
46BF CD752B   00890          CALL   2B75H
46C2 CD4900   00900          CALL   49H
46C5 3E01     00910          LD     A,1
46C7 C31202   00920          JP     212H
46CA CD9744   00930 DIS2     CALL   DISW
46CD 3E2C     00940          LD     A,','
46CF C32A03   00950          JP     32AH
0006          00960 NAME     DEFS   6
46D8 00       00970          NOP
46D9 52       00980 CA1      DEFM   'READY CASSETTE'
46DA 45
46DB 41
46DC 44
46DD 59
46DE 20
46DF 43
46E0 41
46E1 53
46E2 53
46E3 45
46E4 54
46E5 54
46E6 45
46E7 0D       00990          DEFB   13
46E8 00       01000          NOP
46E9 4E       01010 CA2      DEFM   'NAME: '
46EA 41
46EB 4D
46EC 45
46ED 3A
46EE 20
46EF 00       01020          NOP
46F0 2A8945   01030 PUNCH    LD     HL,(N3)
46F3 7C       01040          LD     A,H
46F4 B5       01050          OR     L
46F5 2006     01060          JR     NZ,PUN
46F7 2A8545   01070          LD     HL,(N1)
46FA 228945   01080          LD     (N3),HL
46FD 21E946   01090 PUN      LD     HL,CA2
4700 CD752B   01100          CALL   2B75H
4703 CD6103   01110          CALL   361H
4706 DAE948   01120          JP     C,L1
4709 D7       01130          RST    16
470A 2003     01140          JR     NZ,PU1
470C 217D47   01150          LD     HL,CA3
470F E5       01160 PU1      PUSH   HL
```

```
4710 CDBC46    01170          CALL    CASS
4713 FDE1      01180          POP     IY
4715 CDCA47    01190          CALL    LENGTH
4718 CD8402    01200          CALL    284H
471B 3E55      01210          LD      A,55H
471D CD6402    01220          CALL    264H
4720 0606      01230          LD      B,6
4722 FD7E00    01240 PU2      LD      A,(IY+0)
4725 CD6402    01250          CALL    264H
4728 FD23      01260          INC     IY
472A 10F6      01270          DJNZ    PU2
472C 25        01280 PU3      DEC     H
472D FA3E47    01290          JP      M,PU4
4730 3E3C      01300          LD      A,3CH
4732 CD6402    01310          CALL    264H
4735 AF        01320          XOR     A
4736 CD6402    01330          CALL    264H
4739 CD6547    01340          CALL    PU5
473C 18EE      01350          JR      PU3
473E AF        01360 PU4      XOR     A
473F BD        01370          CP      L
4740 280C      01380          JR      Z,PU6
4742 3E3C      01390          LD      A,3CH
4744 CD6402    01400          CALL    264H
4747 7D        01410          LD      A,L
4748 CD6402    01420          CALL    264H
474B CD6547    01430          CALL    PU5
474E 3E78      01440 PU6      LD      A,78H
4750 CD6402    01450          CALL    264H
4753 3A8945    01460          LD      A,(N3)
4756 CD6402    01470          CALL    264H
4759 3A8A45    01480          LD      A,(N3+1)
475C CD6402    01490          CALL    264H
475F CDF801    01500          CALL    1F8H
4762 C3E948    01510          JP      L1
4765 47        01520 PU5      LD      B,A
4766 7B        01530          LD      A,E
4767 CD6402    01540          CALL    264H
476A 7A        01550          LD      A,D
476B CD6402    01560          CALL    264H
476E 83        01570          ADD     A,E
476F 4F        01580          LD      C,A
4770 1A        01590 PU7      LD      A,(DE)
4771 CD6402    01600          CALL    264H
4774 81        01610          ADD     A,C
4775 4F        01620          LD      C,A
4776 13        01630          INC     DE
4777 10F7      01640          DJNZ    PU7
4779 79        01650          LD      A,C
477A C36402    01660          JP      264H
477D 4E        01670 CA3      DEFM    'NONAME'
477E 4F
477F 4E
4780 41
4781 4D
4782 45
4783 CD9F44    01680 ASC      CALL    CH1
4786 2A8545    01690          LD      HL,(N1)
4789 CD8544    01700 AS1      CALL    DIS1
478C 0630      01710          LD      B,48
478E 7E        01720 AS2      LD      A,(HL)
478F CBBF      01730          RES     7,A
4791 FE20      01740          CP      32
4793 3804      01750          JR      C,AS3
4795 FE80      01760          CP      128
4797 3802      01770          JR      C,AS4
4799 3E2E      01780 AS3      LD      A,'.'
```

```
479B CD2A03    01790 AS4     CALL    32AH
479E CD2046    01800         CALL    CH2
47A1 10EB      01810         DJNZ    AS2
47A3 CD8044    01820         CALL    CR
47A6 CD1146    01830         CALL    KEY
47A9 18DE      01840         JR      AS1
47AB CD9F44    01850 FIND    CALL    CH1
47AE CDD647    01860 F1      CALL    SEARCH
47B1 F5        01870         PUSH    AF
47B2 CDD348    01880         CALL    CH19
47B5 7E        01890         LD      A,(HL)
47B6 CDC548    01900         CALL    CH12
47B9 F1        01910         POP     AF
47BA E2E948    01920         JP      PO,L1
47BD CD1146    01930         CALL    KEY
47C0 18EC      01940         JR      F1
47C2 CD9F44    01950 CH9     CALL    CH1
47C5 1803      01960         JR      LENGTH
47C7 CDD144    01970 CH7     CALL    CHECK
47CA 2A8745    01980 LENGTH  LD      HL,(N2)
47CD ED5B8545  01990         LD      DE,(N1)
47D1 AF        02000         XOR     A
47D2 ED52      02010         SBC     HL,DE
47D4 23        02020         INC     HL
47D5 C9        02030         RET
47D6 CDCA47    02040 SEARCH  CALL    LENGTH
47D9 E5        02050         PUSH    HL
47DA C1        02060         POP     BC
47DB 2A8545    02070         LD      HL,(N1)
47DE 3A8945    02080         LD      A,(N3)
47E1 EDB1      02090         CPIR
47E3 C2E948    02100         JP      NZ,L1
47E6 228545    02110         LD      (N1),HL
47E9 C9        02120         RET
47EA CD9F44    02130 HUNT    CALL    CH1
47ED CDD647    02140 H1      CALL    SEARCH
47F0 F5        02150         PUSH    AF
47F1 2B        02160         DEC     HL
47F2 CDD348    02170         CALL    CH19
47F5 5E        02180         LD      E,(HL)
47F6 23        02190         INC     HL
47F7 56        02200         LD      D,(HL)
47F8 EB        02210         EX      DE,HL
47F9 CD9744    02220         CALL    DISW
47FC CD8044    02230         CALL    CR
47FF F1        02240         POP     AF
4800 E2E948    02250         JP      PO,L1
4803 CD1146    02260         CALL    KEY
4806 18E5      02270         JR      H1
4808 CDBA48    02280 MOVE    CALL    CH8
480B CDCB48    02290         CALL    CH13
480E EDB0      02300         LDIR
4810 C3E948    02310         JP      L1
4813 CDC247    02320 SUM     CALL    CH9
4816 EB        02330         EX      DE,HL
4817 2A8545    02340         LD      HL,(N1)
481A 47        02350         LD      B,A
481B 7E        02360 S1      LD      A,(HL)
481C 80        02370         ADD     A,B
481D 47        02380         LD      B,A
481E 23        02390         INC     HL
481F 1B        02400         DEC     DE
4820 7A        02410         LD      A,D
4821 B3        02420         OR      E
4822 20F7      02430         JR      NZ,S1
4824 C5        02440         PUSH    BC
4825 213248    02450         LD      HL,SU1
4828 CD752B    02460         CALL    2B75H
```

```
482B F1         02470          POP    AF
482C CD6343     02480          CALL   D1
482F C32946     02490          JP     CH3
4832 43         02500 SU1      DEFM   'CHECKSUM: '
4833 48
4834 45
4835 43
4836 4B
4837 53
4838 55
4839 4D
483A 3A
483B 20
483C 00         02510          NOP
483D CDC048     02520 TEST     CALL   CH10
4840 2A8545     02530          LD     HL,(N1)
4843 5E         02540 T1       LD     E,(HL)
4844 AF         02550          XOR    A
4845 77         02560          LD     (HL),A
4846 56         02570          LD     D,(HL)
4847 BA         02580          CP     D
4848 73         02590          LD     (HL),E
4849 280C       02600          JR     Z,T2
484B D5         02610          PUSH   DE
484C CD8544     02620          CALL   DIS1
484F AF         02630          XOR    A
4850 CD9244     02640          CALL   DSP
4853 F1         02650          POP    AF
4854 CDC548     02660          CALL   CH12
4857 5E         02670 T2       LD     E,(HL)
4858 3EFF       02680          LD     A,0FFH
485A 77         02690          LD     (HL),A
485B 56         02700          LD     D,(HL)
485C BA         02710          CP     D
485D 73         02720          LD     (HL),E
485E 280D       02730          JR     Z,T3
4860 D5         02740          PUSH   DE
4861 CD8544     02750          CALL   DIS1
4864 3EFF       02760          LD     A,0FFH
4866 CD9244     02770          CALL   DSP
4869 F1         02780          POP    AF
486A CDC548     02790          CALL   CH12
486D CDDD48     02800 T3       CALL   CH18
4870 18D1       02810          JR     T1
4872 CDC048     02820 VERF     CALL   CH10
4875 CDCB48     02830 V1       CALL   CH13
4878 7E         02840          LD     A,(HL)
4879 EB         02850          EX     DE,HL
487A BE         02860          CP     (HL)
487B 23         02870          INC    HL
487C 13         02880          INC    DE
487D ED538545   02890          LD     (N1),DE
4881 228945     02900          LD     (N3),HL
4884 281B       02910          JR     Z,V2
4886 E5         02920          PUSH   HL
4887 EB         02930          EX     DE,HL
4888 2B         02940          DEC    HL
4889 CD8544     02950          CALL   DIS1
488C 7E         02960          LD     A,(HL)
488D CD9244     02970          CALL   DSP
4890 3E2D       02980          LD     A,'-'
4892 CD2A03     02990          CALL   32AH
4895 CD8D44     03000          CALL   SPA
4898 E1         03010          POP    HL
4899 2B         03020          DEC    HL
489A CD8544     03030          CALL   DIS1
489D 7E         03040          LD     A,(HL)
489E CDC548     03050          CALL   CH12
```

```
48A1 CDDE48    03060 V2      CALL    CH17
48A4 C37548    03070         JP      V1
48A7 CDBA48    03080 EXCH    CALL    CH8
48AA CDCB48    03090         CALL    CH13
48AD C5        03100 E1      PUSH    BC
48AE 46        03110         LD      B,(HL)
48AF EB        03120         EX      DE,HL
48B0 7E        03130         LD      A,(HL)
48B1 70        03140         LD      (HL),B
48B2 EB        03150         EX      DE,HL
48B3 77        03160         LD      (HL),A
48B4 C1        03170         POP     BC
48B5 CDE348    03180         CALL    CH14
48B8 18F3      03190         JR      E1
48BA CDC747    03200 CH8     CALL    CH7
48BD E5        03210 CH11    PUSH    HL
48BE C1        03220         POP     BC
48BF C9        03230         RET
48C0 CDC247    03240 CH10    CALL    CH9
48C3 18F8      03250         JR      CH11
48C5 CD9244    03260 CH12    CALL    DSP
48C8 C38044    03270         JP      CR
48CB 2A8545    03280 CH13    LD      HL,(N1)
48CE ED5B8945  03290         LD      DE,(N3)
48D2 C9        03300         RET
48D3 2B        03310 CH19    DEC     HL
48D4 CD8544    03320         CALL    DIS1
48D7 7E        03330 CH20    LD      A,(HL)
48D8 CD9244    03340         CALL    DSP
48DB 23        03350         INC     HL
48DC C9        03360         RET
48DD 23        03370 CH18    INC     HL
48DE CD1146    03380 CH17    CALL    KEY
48E1 1802      03390         JR      CH16
48E3 13        03400 CH14    INC     DE
48E4 23        03410 CH15    INC     HL
48E5 0B        03420 CH16    DEC     BC
48E6 78        03430         LD      A,B
48E7 B1        03440         OR      C
48E8 C0        03450         RET     NZ
48E9 00        03460 L1      DEFB    0
4300           03470         END     4300H
00000 TOTAL ERRORS

CH15    48E4
CH16    48E5
CH20    48D7
CH11    48BD
CH14    48E3
E1      48AD
EXCH    48A7
CH17    48DE
V2      48A1
V1      4875
VERF    4872
CH18    48DD
T3      486D
T2      4857
T1      4843
CH10    48C0
TEST    483D
SU1     4832
S1      481B
SUM     4813
CH13    48CB
CH8     48BA
MOVE    4808
H1      47ED
```

HUNT	47EA
CH7	47C7
CH9	47C2
CH12	48C5
CH19	48D3
SEARCH	47D6
F1	47AE
FIND	47AB
AS4	479B
AS3	4799
AS2	478E
AS1	4789
ASC	4783
PU7	4770
PU6	474E
PU5	4765
PU4	473E
PU3	472C
PU2	4722
LENGTH	47CA
CA3	477D
PU1	470F
PUN	46FD
PUNCH	46F0
CA1	46D9
DIS2	46CA
CA2	46E9
RA5	4672
RA6	468B
RA4	464F
RA3	464C
RA2	4645
NAME	46D2
RA1	4639
CASS	46BC
READ	462F
L1	48E9
KEY	4611
CH3	4629
CH2	4620
CH1	449F
SPA	448D
D1	4363
DSP	4492
DIS1	4485
PR	44A2
CHECK	44D1
CR	4480
DISW	4497
FLAG	458C
N3	4589
N2	4587
N1	4585

Listing 10-3 Comments

100 - Value from previous part.
110 - Value from previous part.
120 - Value from previous part.
130 - Value from previous part.
140 - Value from previous part.
150 - Value from previous part.
160 - Value from previous part.

170 - Value from previous part.
180 - Value from previous part.
190 - Value from previous part.
200 - Value from previous part.
210 - Value from previous part.
220 - Value from previous part.
230 - Value from previous part.
240 - Value from previous part.
250 - Value from previous part.
260 - Set the starting address to the end of the last part.
270 - Jump.
280 - Zero register A.
290 - Zero the input flag.
300 - Go display the prompt and turn on the cassette.
310 - Go read the leader and find the sync byte.
320 - Go read a byte from the cassette recorder and return with it in register A.
330 - Check to see if the byte read from the cassette is the filename header byte.
340 - Loop till the filename header byte is found.
350 - Load register B with the number of bytes to be read for the filename.
360 - Load register pair HL with the filename storage location.
370 - Go read a byte from the cassette recorder and return with the byte in register A.
380 - Save the byte read from the cassette in register A at the location of the filename storage pointer in register pair HL.
390 - Bump the filename storage pointer in register pair HL.
400 - Loop till all of the bytes in the filename have been read.
410 - Go blink the asterisks on the video display.
420 - Go read a byte from the cassette recorder and return with it in register A.
430 - Check to see if the byte read from the cassette recorder is the jump address header.
440 - Jump if the byte read from the cassette recorder is the jump address header.
450 - Check to see if the byte read from the cassette in register A is the data block header.
460 - Jump if the byte read from the cassette in register A isn't a data block header.
470 - Go read a byte from the cassette recorder and return with it in register A.

480 - Save the number of bytes in the file block in register A in register B.

490 - Go read two bytes from the cassette recorder and return with the data block starting address in register pair HL.

500 - Add the MSB of the data block's starting address in register A to the LSB of the data block's starting address in register L.

510 - Save the combined starting address in register A as the starting checksum in register C.

520 - Load register A with the input flag.

530 - Check to see if the starting address has been saved.

540 - Jump if the program's starting address has already been saved.

550 - Save the starting address in register pair HL.

560 - Load register A with the value to indicate the starting address has been saved.

570 - Save the value in register A in the input flag.

580 - Go read a byte from the cassette recorder and return with it in register A.

590 - Save the value in register A at the location of the memory pointer in register pair HL.

600 - Bump the memory pointer in register pair HL.

610 - Save the memory pointer in register pair HL as the ending address.

620 - Add the current checksum in register C with the value in register A.

630 - Save the updated checksum in register A in register C.

640 - Loop till the data block has been read.

650 - Go read the checksum from the cassette recorder and return with it in register A.

660 - Compare the checksum read from the tape in register A with the computed checksum in register C.

670 - Jump if the checksum read from the tape in register A matches the checksum computed in register C.

680 - Load register A with a C character.

690 - Go display the C character in register A on the video display.

700 - Jump.

710 - Go read the execution address from the cassette recorder and return with it in register pair HL.

720 - Save the execution address in register pair HL.

730 - Go turn off the cassette recorder.

740 - Load register pair HL with the starting address for the message to be displayed.
750 - Go display the message.
760 - Load register pair HL with the starting address for the filename.
770 - Go display the filename.
780 - Go display a carriage return.
790 - Load register pair HL with the starting address.
800 - Go display the program's starting address in register pair HL.
810 - Load register pair HL with the program's ending address.
820 - Decrement the program's ending address in register pair HL.
830 - Go display the program's ending address in register pair HL.
840 - Load register pair HL with the program's execution address.
850 - Go display the program's execution address in register pair HL.
860 - Go display a carriage return.
870 - Jump.
880 - Load register pair HL with the starting address of the cassette prompting message.
890 - Go display the message.
900 - Go wait till a key is pressed.
910 - Load register A with the drive number to turn on.
920 - Go start the cassette motor.
930 - Go display the value in register pair HL.
940 - Load register A with a comma character.
950 - Go display a comma.
960 - The program's name will be stored here.
970 - The program's name message terminator is stored here.
980 - The cassette prompting message is stored here.
990 - This will display a carriage return as part of the cassette prompting message.
1000 - The cassette prompting message terminator is stored here.
1010 - The NAME message is stored here.
1020 - The NAME message terminator is stored here.
1030 - Load register pair HL with the program's execution address.
1040 - Load register A with the MSB of the program's execution address in register H.

1050 - Combine the MSB of the program's execution address in register A with the LSB of the program's execution address in register L.
1060 - Jump if the execution address in register pair HL isn't equal to zero.
1070 - Load register pair HL with the program's starting address.
1080 - Save the program's starting address in register pair HL as the program's execution address.
1090 - Load register pair HL with the starting address of the NAME message.
1100 - Go display the NAME message.
1110 - Go get the keyboard input.
1120 - Jump if the BREAK key was pressed.
1130 - Go bump the input buffer pointer in register pair HL till it points to the first character in the input.
1140 - Jump if a name was input.
1150 - Load register pair HL with the starting address for the NONAME program name.
1160 - Save the starting address for the program name in register pair HL on the stack.
1170 - Go prompt for the cassette and turn on the cassette motor.
1180 - Get the program name's starting address from the stack and put it in register pair IY.
1190 - Go figure the program's length and return with it in register pair HL.
1200 - Go write the leader and the sync byte.
1210 - Load register A with the filename header byte.
1220 - Go write the filename header byte on the cassette recorder.
1230 - Load register B with the length of the filename to write on the cassette recorder.
1240 - Load register A with the character at the filename pointer in register pair IY.
1250 - Go write the filename character in register A on the cassette recorder.
1260 - Bump the filename pointer in register pair IY.
1270 - Loop till the filename has been completely written on the cassette.
1280 - Decrement the MSB of the program's length in register H.
1290 - Jump if this is the last data block to be written.
1300 - Load register A with the data block file header.
1310 - Go write the data block file header byte on the cassette recorder.

1320 - Load register A with the data block's length.
1330 - Go write the data block's length on the cassette recorder.
1340 - Go write the data block.
1350 - Loop till all but the last data block has been written.
1360 - Zero register A.
1370 - Check to see if the LSB of the program's length in register L is zero.
1380 - Jump if the LSB of the program's length is zero.
1390 - Load register A with the data block header byte.
1400 - Go write the data block header byte on the cassette.
1410 - Load register A with the data block's length in register L.
1420 - Go write the data block's length on the cassette recorder.
1430 - Go write the data block on the cassette recorder.
1440 - Load register A with the execution address header byte.
1450 - Go write the execution address header byte on the cassette recorder.
1460 - Load register A with the LSB of the execution address.
1470 - Go write the execution address's LSB on the cassette recorder.
1480 - Load register A with the MSB of the execution address.
1490 - Go write the MSB of the execution address on the cassette recorder.
1500 - Go turn off the cassette recorder.
1510 - Jump.
1520 - Load register B with the data block's length in register A.
1530 - Load register A with the data block's starting address in register E.
1540 - Go write the LSB of the data block's starting address on the cassette recorder.
1550 - Load register A with the MSB of the data block's starting address in register D.
1560 - Go write the MSB of the data block's starting address on the cassette recorder.
1570 - Add the LSB of the data block's starting address in register E to the MSB of the data block's starting address in register A.
1580 - Load register C with the starting checksum in register A.
1590 - Load register A with the byte at the location of the memory pointer in register pair DE.
1600 - Go write the byte in register A on the cassette recorder.
1610 - Add the current checksum in register C to the value in register A.

1620 - Save the updated checksum in register A in register C.
1630 - Bump the memory pointer in register pair DE.
1640 - Loop till the data block has been completely written on the cassette recorder.
1650 - Load register A with the checksum in register C.
1660 - Go write the checksum on the cassette recorder.
1670 - The NONAME program file name is stored here.
1680 - Go check the input values.
1690 - Load register pair HL with the starting address.
1700 - Go display the memory pointer in register pair HL.
1710 - Load register B with the number of characters to be displayed.
1720 - Load register A with the value at the location of the memory pointer in register pair HL.
1730 - Clear bit-7 of the character to be displayed in register A.
1740 - Check to see if the character in register A is a control code.
1750 - Jump if the character in register A is a control code.
1760 - Check to see if the character in register A is greater than 80H.
1770 - Jump if the character in register A is less than 80H.
1780 - Load register A with a . character.
1790 - Go display the character in register A.
1800 - Go check to see if the memory pointer in register pair HL is the same as the ending address.
1810 - Loop till 48 characters have been displayed.
1820 - Go display a carriage return.
1830 - Go check to see if a key has been pressed.
1840 - Jump till all of the memory locations have been displayed.
1850 - Go check the input values.
1860 - See if the character can be found.
1870 - Save the flags on the stack.
1880 - Go display the memory location and the value found.
1890 - Load register A with the next byte after the one found.
1900 - Go display the byte in register A.
1910 - Get the flags from the stack.
1920 - Jump if this is the end of the search.
1930 - Go check to see if a key is pressed.
1940 - Jump till the end of the search.
1950 - Go check the input values.
1960 - Jump.
1970 - Go check the input values.
1980 - Load register pair HL with the ending address.

1990 - Load register pair DE with the starting address.
2000 - Clear the Carry flag.
2010 - Subtract the starting address in register pair DE from the ending address in register pair HL.
2020 - Bump the length in register pair HL.
2030 - Return.
2040 - Go figure the length and return with the length in register pair HL.
2050 - Save the length in register pair HL on the stack.
2060 - Get the length from the stack and put it in register pair BC.
2070 - Load register pair HL with the starting address.
2080 - Load register A with the value to be found.
2090 - Go see if the value in register A can be found.
2100 - Jump if the value in register A can't be found.
2110 - Save the next value to be searched for in register pair HL.
2120 - Return.
2130 - Go check the input values.
2140 - Go check to see if the value can be found.
2150 - Save the flags on the stack.
2160 - Decrement the memory pointer in register pair HL.
2170 - Go display the memory location and the value at the location of the memory pointer in register pair HL.
2180 - Load register E with the LSB of the value at the location of the memory pointer in register pair HL.
2190 - Bump the memory pointer in register pair HL.
2200 - Load register D with the MSB of the value at the location of the memory pointer in register pair HL.
2210 - Exchange the memory pointer in register pair HL with the value in register pair DE.
2220 - Go display the value in register pair HL.
2230 - Go display a carriage return.
2240 - Get the flags from the stack.
2250 - Jump if this is the end of the search.
2260 - Go check to see if a key has been pressed.
2270 - Jump if the search hasn't been completed.
2280 - Go get the length of the portion of memory to be moved and return with it in register pair BC.
2290 - Go get the starting address of the section of memory to be moved in register pair HL and the new starting address in register pair DE.
2300 - Go move the memory.

2310 - Jump.
2320 - Go get the length of the section of memory to be checksummed.
2330 - Load register pair DE with the length of the section of memory to be checksummed.
2340 - Load register pair HL with the starting address.
2350 - Load register B with the starting checksum.
2360 - Load register A with the value at the location of the memory pointer in register pair HL.
2370 - Add the current checksum in register B to the value in register A.
2380 - Load register B with the updated checksum in register A.
2390 - Bump the memory pointer in register pair HL.
2400 - Decrement the length in register pair DE.
2410 - Load register A with the MSB of the length in register D.
2420 - Combine the LSB of the length in register E with the MSB of the length in register A.
2430 - Jump if this isn't the end of the section of memory to be checksummed.
2440 - Save the checksum in register B on the stack.
2450 - Load register pair HL with the starting address of the message to be displayed.
2460 - Go display the message.
2470 - Get the checksum from the stack and put it in register A.
2480 - Go display the checksum in register A.
2490 - Jump.
2500 - The checksum message is stored here.
2510 - The checksum message terminator is stored here.
2520 - Go check the input values and get the length of the section of memory to be tested.
2530 - Load register pair HL with the starting address.
2540 - Load register E with the value at the location of the memory pointer in register pair HL.
2550 - Zero register A.
2560 - Save the value in register A at the location of the memory pointer in register pair HL.
2570 - Load register D with the value at the location of the memory pointer in register pair HL.
2580 - Compare the value in register D with the expected value in register A.
2590 - Save the original value in register E at the location of the

memory pointer in register pair HL.

2600 - Jump if the value in register D is the same as the value in register A.

2610 - Save the value in register D on the stack.

2620 - Go display the value of the memory pointer in register pair HL.

2630 - Zero register A.

2640 - Go display the expected value in register A.

2650 - Get the actual value from the stack and put it in register A.

2660 - Go display the value in register A and a carriage return.

2670 - Load register E with the value at the location of the memory pointer in register pair HL.

2680 - Load register A with the value to be tested.

2690 - Save the value in register A at the location of the memory pointer in register pair HL.

2700 - Load register D with the value at the location of the memory pointer in register pair HL.

2710 - Check to see if the value in register D matches the expected value in register A.

2720 - Save the original value in register E at the location of the memory pointer in register pair HL.

2730 - Jump if the value in register D matches the expected value in register A.

2740 - Save the value in register D on the stack.

2750 - Go display the memory pointer in register pair HL.

2760 - Load register A with the expected value.

2770 - Go display the expected value in register A.

2780 - Get the actual value from the stack and put it in register A.

2790 - Go display the actual value in register A and a carriage return.

2800 - Go check to see if this is the last location to be checked.

2810 - Jump till all of the locations have been tested.

2820 - Go check the input values and return with the length in register pair BC.

2830 - Load register pair HL with the first starting address and register pair DE with the second starting address.

2840 - Load register A with the value at the location of the memory pointer in register pair HL.

2850 - Exchange the memory pointer in register pair HL with the second memory pointer in register pair DE.

2860 - Compare the value in register A with the value at the location of the second memory pointer in register pair HL.

2870 - Bump the second memory pointer in register pair HL.
2880 - Bump the first memory pointer in register pair DE.
2890 - Save the first memory pointer in register pair DE.
2900 - Save the second memory pointer in register pair HL.
2910 - Jump if the two values in memory match.
2920 - Save the second memory pointer in register pair HL on the stack.
2930 - Load register pair HL with the first memory pointer.
2940 - Decrement the first memory pointer in register pair HL.
2950 - Go display the value of the first memory pointer in register pair HL.
2960 - Load register A with the value at the location of the first memory pointer in register pair HL.
2970 - Go display the value in register A.
2980 - Load register A with a - character.
2990 - Go display the character in register A.
3000 - Go display a space.
3010 - Get the second memory pointer from the stack and put it in register pair HL.
3020 - Decrement the second memory pointer in register pair HL.
3030 - Go display the second memory pointer in register pair HL.
3040 - Load register A with the value at the location of the memory pointer in register pair HL.
3050 - Go display the value in register A and a carriage return.
3060 - Go check to see if this is the last location to be checked.
3070 - Loop till all of the locations have been checked.
3080 - Go check the input values and return with the length in register pair BC.
3090 - Go load register pair HL with the first starting address and register pair DE with the second starting address.
3100 - Save the length in register pair BC on the stack.
3110 - Load register B with the value at the location of the first memory pointer in register pair HL.
3120 - Exchange the memory pointer in register pair HL with the second memory pointer in register pair DE.
3130 - Load register A with the value at the location of the second memory pointer in register pair HL.
3140 - Save the value in register B at the location of the second memory pointer in register pair HL.
3150 - Exchange the second memory pointer in register pair HL with the first memory pointer in register pair DE.
3160 - Save the value in register A at the location of the first

memory pointer in register pair HL.
3170 - Get the length from the stack and put it in register pair BC.
3180 - Go bump the memory pointers in register pair HL and register pair DE and check to see if all locations have been exchanged.
3190 - Loop till all of the locations have been exchanged.
3200 - Go check the input values and return with the length in register pair HL.
3210 - Save the length in register pair HL on the stack.
3220 - Get the length from the stack and put it in register pair BC.
3230 - Return.
3240 - Go check the input values and return with the length in register pair HL.
3250 - Jump.
3260 - Go display the value in register A.
3270 - Go display a carriage return.
3280 - Load register pair HL with the first value that was input.
3290 - Load register pair DE with the third value that was input.
3300 - Return.
3310 - Decrement the value in register pair HL.
3320 - Go display the value in register pair HL.
3330 - Load register A with the value at the location of the memory pointer in register pair HL.
3340 - Go display the value in register A.
3350 - Bump the memory pointer in register pair HL.
3360 - Return.
3370 - Bump the value in register pair HL.
3380 - Go check to see if a key was pressed.
3390 - Jump.
3400 - Bump the value in register pair DE.
3410 - Bump the value in register pair HL.
3420 - Decrement the value in register pair BC.
3430 - Load register A with the value in register B.
3440 - Combine the value in register C with the value in register A.
3450 - Return if register BC doesn't equal zero.
3460 - Link with the next part.
3470 - End of the program.

Listing 10-4

```
4585          00100 N1      EQU     4585H
4587          00110 N2      EQU     4587H
```

```
4589              00120 N3       EQU      4589H
430F              00130 HEX      EQU      430FH
458D              00140 FL2      EQU      458DH
4347              00150 DISP     EQU      4347H
4383              00160 GO       EQU      4383H
4387              00170 EDIT     EQU      4387H
43B2              00180 ZERO     EQU      43B2H
43C4              00190 INP      EQU      43C4H
43D3              00200 OUP      EQU      43D3H
43DF              00210 MATH     EQU      43DFH
44E4              00220 BRK      EQU      44E4H
48A7              00230 EXCH     EQU      48A7H
4872              00240 VERF     EQU      4872H
483D              00250 TEST     EQU      483DH
4813              00260 SUM      EQU      4813H
4808              00270 MOVE     EQU      4808H
47EA              00280 HUNT     EQU      47EAH
47AB              00290 FIND     EQU      47ABH
46F0              00300 PUNCH    EQU      46F0H
462F              00310 READ     EQU      462FH
4783              00320 ASC      EQU      4783H
48BA              00330 CH8      EQU      48BAH
48CB              00340 CH13     EQU      48CBH
44A9              00350 LO       EQU      44A9H
46CA              00360 DIS2     EQU      46CAH
4497              00370 DISW     EQU      4497H
4480              00380 CR       EQU      4480H
48E9              00390          ORG      48E9H
48E9 310043       00400 L1       LD       SP,4300H
48EC 210000       00410          LD       HL,0
48EF 228545       00420          LD       (N1),HL
48F2 228745       00430          LD       (N2),HL
48F5 228945       00440          LD       (N3),HL
48F8 AF           00450          XOR      A
48F9 329C40       00460          LD       (409CH),A
48FC 3E2A         00470          LD       A,'*'
48FE CD2A03       00480          CALL     32AH
4901 CD6103       00490          CALL     361H
4904 38E3         00500          JR       C,L1
4906 D7           00510          RST      16
4907 F5           00520          PUSH     AF
4908 D7           00530          RST      16
4909 2B           00540          DEC      HL
490A CD0F43       00550          CALL     HEX
490D ED538545     00560          LD       (N1),DE
4911 CD0F43       00570          CALL     HEX
4914 ED538745     00580          LD       (N2),DE
4918 CD0F43       00590          CALL     HEX
491B ED538945     00600          LD       (N3),DE
491F 222F4A       00610          LD       (N5),HL
4922 F1           00620          POP      AF
4923 216D4A       00630          LD       HL,TABLE
4926 0617         00640          LD       B,23
4928 4E           00650 L2       LD       C,(HL)
4929 B9           00660          CP       C
492A 2006         00670          JR       NZ,L3
492C 23           00680          INC      HL
492D 5E           00690          LD       E,(HL)
492E 23           00700          INC      HL
492F 56           00710          LD       D,(HL)
4930 EB           00720          EX       DE,HL
4931 E9           00730          JP       (HL)
4932 23           00740 L3       INC      HL
4933 23           00750          INC      HL
4934 23           00760          INC      HL
4935 10F1         00770          DJNZ     L2
4937 18B0         00780          JR       L1
4939 3E01         00790 PRINT    LD       A,1
```

```
493B 328D45          LD      (FL2),A
493E 18A9            JR      L1
4940 AF       CRT    XOR     A
4941 328D45          LD      (FL2),A
4944 18A3     EXIT   JR      L1
4946 CD2537   ERR2   CALL    3725H
4949 CD2A30   ERROR  CALL    302AH
494C 18F6            JR      EXIT
494E 2A2F4A   SAVE   LD      HL,(N5)
4951 CD0F43          CALL    HEX
4954 7B              LD      A,E
4955 F5              PUSH    AF
4956 CD1B4A          CALL    ESF
4959 F1              POP     AF
495A CD254A          CALL    FILEN
495D F5              PUSH    AF
495E 213F4A          LD      HL,SA2
4961 CD752B          CALL    2B75H
4964 CDBA48          CALL    CH8
4967 F1              POP     AF
4968 CDCB48          CALL    CH13
496B CD0C30          CALL    300CH
496E 20D9     SAV1   JR      NZ,ERROR
4970 21494A          LD      HL,SA3
4973 CD752B          CALL    2B75H
4976 18CC            JR      EXIT
4978 CD1B4A   NEW    CALL    ESF
497B 3A8545          LD      A,(N1)
497E CD254A          CALL    FILEN
4981 F5              PUSH    AF
4982 214F4A          LD      HL,SA4
4985 CD752B          CALL    2B75H
4988 F1              POP     AF
4989 67              LD      H,A
498A CD5F32          CALL    325FH
498D F5              PUSH    AF
498E E6B7            AND     0B7H
4990 20B7            JR      NZ,ERROR
4992 CDAF0F          CALL    0FAFH
4995 21594A          LD      HL,SA5
4998 CD752B          CALL    2B75H
499B F1              POP     AF
499C 18D0            JR      SAV1
499E CD1B4A   LOAD   CALL    ESF
49A1 3A8545          LD      A,(N1)
49A4 CD254A          CALL    FILEN
49A7 F5              PUSH    AF
49A8 21624A          LD      HL,SA6
49AB CD752B          CALL    2B75H
49AE F1              POP     AF
49AF 67              LD      H,A
49B0 2EFF            LD      L,0FFH
49B2 B7              OR      A
49B3 2001            JR      NZ,LO2
49B5 6F              LD      L,A
49B6 CD3437   LO2    CALL    3734H
49B9 CD4936   LO3    CALL    3649H
49BC C24649   ERR3   JP      NZ,ERR2
49BF 7A              LD      A,D
49C0 B7              OR      A
49C1 28F6            JR      Z,LO3
49C3 FAB949          JP      M,LO3
49C6 94              SUB     H
49C7 A5              AND     L
49C8 20EF            JR      NZ,LO3
49CA CD0B37          CALL    370BH
49CD 20ED            JR      NZ,ERR3
```

```
49CF 62       01470          LD     H,D
49D0 E5       01480          PUSH   HL
49D1 DDE1     01490          POP    IX
49D3 228545   01500          LD     (N1),HL
49D6 CD0B37   01510          CALL   370BH
49D9 20E1     01520          JR     NZ,ERR3
49DB 62       01530          LD     H,D
49DC E5       01540          PUSH   HL
49DD FDE1     01550          POP    IY
49DF 228945   01560          LD     (N3),HL
49E2 CD0B37   01570          CALL   370BH
49E5 20D5     01580          JR     NZ,ERR3
49E7 62       01590          LD     H,D
49E8 E5       01600          PUSH   HL
49E9 D1       01610          POP    DE
49EA 228745   01620          LD     (N2),HL
49ED CDA536   01630          CALL   36A5H
49F0 20CA     01640          JR     NZ,ERR3
49F2 CD2537   01650          CALL   3725H
49F5 21494A   01660          LD     HL,SA3
49F8 CD752B   01670          CALL   2B75H
49FB CDA944   01680          CALL   LO
49FE 1B       01690          DEC    DE
49FF 19       01700          ADD    HL,DE
4A00 228745   01710          LD     (N2),HL
4A03 2A8545   01720          LD     HL,(N1)
4A06 CDCA46   01730          CALL   DIS2
4A09 2A8745   01740          LD     HL,(N2)
4A0C CDCA46   01750          CALL   DIS2
4A0F 2A8945   01760          LD     HL,(N3)
4A12 CD9744   01770          CALL   DISW
4A15 CD8044   01780          CALL   CR
4A18 C34449   01790          JP     EXIT
4A1B 216C4A   01800 ESF      LD     HL,DRIVE
4A1E 36F0     01810          LD     (HL),0F0H
4A20 2B       01820          DEC    HL
4A21 22B140   01830          LD     (40B1H),HL
4A24 C9       01840          RET
4A25 B7       01850 FILEN    OR     A
4A26 CA4449   01860          JP     Z,EXIT
4A29 FE64     01870          CP     100
4A2B D24449   01880          JP     NC,EXIT
4A2E C9       01890          RET
4A2F 0000     01900 N5       DEFW   0
4A31 46       01910 SA1      DEFM   'FILE NUMBER: '
4A32 49
4A33 4C
4A34 45
4A35 20
4A36 4E
4A37 55
4A38 4D
4A39 42
4A3A 45
4A3B 52
4A3C 3A
4A3D 20
4A3E 00       01920          NOP
4A3F 57       01930 SA2      DEFM   'WRITING..'
4A40 52
4A41 49
4A42 54
4A43 49
4A44 4E
4A45 47
4A46 2E
4A47 2E
```

```
4A48 00         01940             NOP
4A49 44         01950 SA3         DEFM     'DONE'
4A4A 4F
4A4B 4E
4A4C 45
4A4D 0D         01960             DEFB     13
4A4E 00         01970             NOP
4A4F 45         01980 SA4         DEFM     'ERASING..'
4A50 52
4A51 41
4A52 53
4A53 49
4A54 4E
4A55 47
4A56 2E
4A57 2E
4A58 00         01990             NOP
4A59 20         02000 SA5         DEFM     ' BYTES..'
4A5A 42
4A5B 59
4A5C 54
4A5D 45
4A5E 53
4A5F 2E
4A60 2E
4A61 00         02010             NOP
4A62 52         02020 SA6         DEFM     'READING..'
4A63 45
4A64 41
4A65 44
4A66 49
4A67 4E
4A68 47
4A69 2E
4A6A 2E
4A6B 00         02030             NOP
4A6C 00         02040 DRIVE       NOP
4A6D 44         02050 TABLE       DEFB     'D'
4A6E 4743       02060             DEFW     DISP
4A70 47         02070             DEFB     'G'
4A71 8343       02080             DEFW     GO
4A73 45         02090             DEFB     'E'
4A74 8743       02100             DEFW     EDIT
4A76 5A         02110             DEFB     'Z'
4A77 B243       02120             DEFW     ZERO
4A79 49         02130             DEFB     'I'
4A7A C443       02140             DEFW     INP
4A7C 4F         02150             DEFB     'O'
4A7D D343       02160             DEFW     OUP
4A7F 42         02170             DEFB     'B'
4A80 DF43       02180             DEFW     MATH
4A82 40         02190             DEFB     '@'
4A83 E444       02200             DEFW     BRK
4A85 52         02210             DEFB     'R'
4A86 2F46       02220             DEFW     READ
4A88 50         02230             DEFB     'P'
4A89 F046       02240             DEFW     PUNCH
4A8B 41         02250             DEFB     'A'
4A8C 8347       02260             DEFW     ASC
4A8E 46         02270             DEFB     'F'
4A8F AB47       02280             DEFW     FIND
4A91 48         02290             DEFB     'H'
4A92 EA47       02300             DEFW     HUNT
4A94 4D         02310             DEFB     'M'
4A95 0848       02320             DEFW     MOVE
4A97 51         02330             DEFB     'Q'
4A98 1348       02340             DEFW     SUM
```

```
4A9A 56         02350          DEFB    'V'
4A9B 7248       02360          DEFW    VERF
4A9D 58         02370          DEFB    'X'
4A9E A748       02380          DEFW    EXCH
4AA0 54         02390          DEFB    'T'
4AA1 3D48       02400          DEFW    TEST
4AA3 4C         02410          DEFB    'L'
4AA4 3949       02420          DEFW    PRINT
4AA6 43         02430          DEFB    'C'
4AA7 4049       02440          DEFW    CRT
4AA9 21         02450          DEFB    '!'
4AAA 7849       02460          DEFW    NEW
4AAC 22         02470          DEFB    '"'
4AAD 4E49       02480          DEFW    SAVE
4AAF 23         02490          DEFB    '#'
4AB0 9E49       02500          DEFW    LOAD
4300            02510          END     4300H
00000 TOTAL ERRORS

SA1      4A31
DRIVE    4A6C
ERR3     49BC
LO3      49B9
LO2      49B6
SA6      4A62
LOAD     499E
SA5      4A59
SA4      4A4F
NEW      4978
SA3      4A49
SAV1     496E
SA2      4A3F
FILEN    4A25
ESF      4A1B
SAVE     494E
ERROR    4949
ERR2     4946
EXIT     4944
CRT      4940
PRINT    4939
L3       4932
L2       4928
TABLE    4A6D
N5       4A2F
L1       48E9
CR       4480
DISW     4497
DIS2     46CA
LO       44A9
CH13     48CB
CH8      48BA
ASC      4783
READ     462F
PUNCH    46F0
FIND     47AB
HUNT     47EA
MOVE     4808
SUM      4813
TEST     483D
VERF     4872
EXCH     48A7
BRK      44E4
MATH     43DF
OUP      43D3
INP      43C4
ZERO     43B2
EDIT     4387
```

```
GO      4383
DISP    4347
FL2     458D
HEX     430F
N3      4589
N2      4587
N1      4585
```

Listing 10-4 Comments

100 - Value from previous part.
110 - Value from previous part.
120 - Value from previous part.
130 - Value from previous part.
140 - Value from previous part.
150 - Value from previous part.
160 - Value from previous part.
170 - Value from previous part.
180 - Value from previous part.
190 - Value from previous part.
200 - Value from previous part.
210 - Value from previous part.
220 - Value from previous part.
230 - Value from previous part.
240 - Value from previous part.
250 - Value from previous part.
260 - Value from previous part.
270 - Value from previous part.
280 - Value from previous part.
290 - Value from previous part.
300 - Value from previous part.
310 - Value from previous part.
320 - Value from previous part.
330 - Value from previous part.
340 - Value from previous part.
350 - Value from previous part.
360 - Value from previous part.
370 - Value from previous part.
380 - Value from previous part.
390 - Set the starting address to the end of the last part.
400 - Set the stack pointer.
410 - Load register pair HL with zero.
420 - Save the value in register pair HL as the first value inpu
430 - Save the value in register pair HL as the second value inpu
440 - Save the value in register pair HL as the third value input
450 - Zero register A.

460 - Set the current output device to the video display.
470 - Load register A with a * character.
480 - Go display the character in register A.
490 - Go get the input from the keyboard.
500 - Jump if the BREAK key was pressed.
510 - Bump the input buffer pointer in register pair HL till it points to the first character.
520 - Save the character at the location of the input buffer pointer in register A on the stack.
530 - Bump the input buffer pointer in register pair HL till it points to the next character.
540 - Decrement the input buffer pointer in register pair HL.
550 - Go evaluate the first input value.
560 - Save the value in register pair DE as the first value input.
570 - Go evaluate the second input value.
580 - Save the value in register pair DE as the second value input.
590 - Go evaluate the third value input.
600 - Save the value in register pair DE as the third value input.
610 - Save the input buffer pointer in register pair HL.
620 - Get the command from the stack and put it in register A.
630 - Load register pair HL with the starting address of the command jump table.
640 - Load register B with the number of commands in the jump table.
650 - Load register C with the character at the location of the command jump table pointer in register pair HL.
660 - Check to see if the command in register C matches the command in register A.
670 - Jump if the command in register C doesn't match the command in register A.
680 - Bump the command jump table pointer in register pair HL.
690 - Load register E with the LSB of the command's jump address.
700 - Bump the command jump table pointer in register pair HL.
710 - Load register D with the MSB of the command's jump address.
720 - Load register pair HL with the jump address in register pair DE.
730 - Jump to the command's jump address in register pair HL.
740 - Bump the command jump table pointer in register pair HL.
750 - Bump the command jump table pointer in register pair HL.
760 - Bump the command jump table pointer in register pair HL.

770 - Loop if all of the commands in the jump table haven't been checked.
780 - Jump if the command input isn't a legal command.
790 - Load register A with the printer device type code.
800 - Save the printer device type code in register A.
810 - Jump.
820 - Load register A with the video device type code.
830 - Save the video device type code in register A.
840 - Jump.
850 - Go turn off the stringy floppy.
860 - Go display the stringy floppy error message, if any.
870 - Jump.
880 - Load register pair HL with the input buffer pointer.
890 - Go evaluate the fourth value input.
900 - Load register A with the file number in register E.
910 - Save the file number in register A on the stack.
920 - Set the current stringy floppy drive number to drive 0.
930 - Get the file number from the stack and put it in register A.
940 - Go check to see if this is a legitimate file number in register A.
950 - Save the file number in register A on the stack.
960 - Load register pair HL with the starting address of the message to be displayed.
970 - Go display the message.
980 - Go figure the length of the program to be saved and return with it in register pair BC.
990 - Get the file number from the stack and put it in register A.
1000 - Go get the starting address for the program in register pair HL and the execution address for the program in register pair DE.
1010 - Go save the program.
1020 - Jump if a stringy floppy error occurred.
1030 - Load register pair HL with the starting address for the message to be displayed.
1040 - Go display the message.
1050 - Jump.
1060 - Go set the current stringy floppy drive number to drive 0.
1070 - Load register A with the file number for the start of the wafer certification.
1080 - Go check to see if the file number in register A is a legitimate file number.
1090 - Save the file number in register A on the stack.

1100 - Load register pair HL with the starting address of the message to be displayed.
1110 - Go display the message.
1120 - Get the file number from the stack and put it in register A.
1130 - Load register H with the file number in register A.
1140 - Go certify the wafer.
1150 - Save the flags on the stack.
1160 - Check to see if an error occurred.
1170 - Jump if there was an error.
1180 - Go display the number of bytes certified.
1190 - Load register pair HL with the starting address for the message to be displayed.
1200 - Go display the message.
1210 - Get the flags from the stack.
1220 - Jump.
1230 - Go set the current stringy floppy drive number to drive 0.
1240 - Load register A with the file number to be loaded.
1250 - Go check to see if the file number in register A is a legitimate file number.
1260 - Save the file number in register A on the stack.
1270 - Load register pair HL with the starting address of the message to be displayed.
1280 - Go display the message.
1290 - Get the file number from the stack and put it in register A.
1300 - Load register H with the file number in register A.
1310 - Load register L with a −1.
1320 - Check to see if the file number in register A is equal to zero.
1330 - Jump if the file number in register A isn't equal to zero.
1340 - Zero register L.
1350 - Go turn on the stringy floppy.
1360 - Go position the wafer to the next file.
1370 - Jump if an error occurred.
1380 - Load register A with the value in register D.
1390 - Set the flags.
1400 - Jump if this isn't the start of a file.
1410 - Jump if this isn't the start of a file.
1420 - Subtract the file number in register H from the file number read from the stringy floppy in register A.
1430 - Mask the adjusted file number in register A with the value in register L.
1440 - If this isn't the correct file number then jump.
1450 - Go read the starting address from the stringy floppy and

return with the LSB of the starting address in register L and the MSB of the starting address in register D.

1460 - Jump if a stringy floppy error occurred.

1470 - Load register H with the MSB of the starting address in register D.

1480 - Save the starting address in register pair HL on the stack.

1490 - Get the starting address from the stack and put it in register pair IX.

1500 - Save the starting address in register pair HL.

1510 - Go read the execution address from the stringy floppy and return with the LSB of the execution address in register L and the MSB of the execution address in register D.

1520 - Jump if a stringy floppy error occurred.

1530 - Load register H with the MSB of the execution address in register D.

1540 - Save the execution address in register pair HL on the stack.

1550 - Get the execution address from the stack and put it in register pair IY.

1560 - Save the execution address in register pair HL.

1570 - Go read the length of the program from the stringy floppy and return with the LSB of the program's length in register L and the MSB of the program's length in register D.

1580 - Jump if a stringy floppy error occurred.

1590 - Load register H with the MSB of the program's length in register D.

1600 - Get the program's length from the stack and put it in register pair DE.

1620 - Save the program's length in register pair HL.

1630 - Go read the program from the stringy floppy.

1640 - Jump if a stringy floppy error occurred.

1650 - Go turn off the stringy floppy.

1660 - Load register pair HL with the starting address of the message to be displayed.

1670 - Go display the message.

1680 - Go load register pair HL with the starting address and register pair DE with the program's length.

1690 - Decrement the program's length in register pair DE.

1700 - Add the program's length in register pair DE to the program's starting address in register pair HL.

1710 - Save the program's ending address in register pair HL.

1720 - Load register pair HL with the program's starting address.

1730 - Go display the program's starting address in register pair HL.
1740 - Load register pair HL with the program's ending address.
1750 - Go display the program's ending address in register pair HL.
1760 - Load register pair HL with the program's execution address.
1770 - Go display the program's execution address in register pair HL.
1780 - Go display a carriage return.
1790 - Jump.
1800 - Load register pair HL with the current stringy floppy drive number storage location.
1810 - Set the current stringy floppy drive number to drive 0.
1820 - Decrement the memory pointer in register pair HL.
1830 - Save the memory pointer in register pair HL as the current BASIC memory size pointer.
1840 - Return.
1850 - Check to see if the file number in register A is equal to zero.
1860 - Jump if the file number in register A is equal to zero.
1870 - Check to see if the file number in register A is greater than 99.
1880 - Jump if the file number in register A is greater than 99.
1890 - Return.
1900 - The current input buffer pointer is stored here.
1910 - Message storage location.
1920 - Message terminator.
1930 - The WRITING message is stored here.
1940 - The WRITING message terminator is stored here.
1950 - The DONE message is stored here.
1960 - This will display a carriage return as part of the DONE message.
1970 - The DONE message terminator.
1980 - The ERASING message is stored here.
1990 - The ERASING message terminator is stored here.
2000 - The BYTES message is stored here.
2010 - The BYTES message terminator is stored here.
2020 - The READING message is stored here.
2030 - The READING message terminator is stored here.
2040 - The current stringy floppy drive number is stored here.
2050 - The D command character is stored here.

2060 - The D command jump address is stored here.
2070 - The G command character is stored here.
2080 - The G command jump address is stored here.
2090 - The E command character is stored here.
2100 - The E command jump address is stored here.
2110 - The Z command character is stored here.
2120 - The Z command jump address is stored here.
2130 - The I command character is stored here.
2140 - The I command jump address is stored here.
2150 - The O command character is stored here.
2160 - The O command jump address is stored here.
2170 - The B command character is stored here.
2180 - The B command jump address is stored here.
2190 - The @ command character is stored here.
2200 - The @ command jump address is stored here.
2210 - The R command character is stored here.
2220 - The R command jump address is stored here.
2230 - The P command character is stored here.
2240 - The P command jump address is stored here.
2250 - The A command character is stored here.
2260 - The A command jump address is stored here.
2270 - The F command character is stored here.
2280 - The F command jump address is stored here.
2290 - The H command character is stored here.
2300 - The H command jump address is stored here.
2310 - The M command character is stored here.
2320 - The M command jump address is stored here.
2330 - The Q command character is stored here.
2340 - The Q command jump address is stored here.
2350 - The V command character is stored here.
2360 - The V command jump address is stored here.
2370 - The X command character is stored here.
2380 - The X command jump address is stored here.
2390 - The T command character is stored here.
2400 - The T command jump address is stored here.
2410 - The L command character is stored here.
2420 - The L command jump address is stored here.
2430 - The C command character is stored here.
2440 - The C command jump address is stored here.
2450 - The ! command character is stored here.
2460 - The ! command jump address is stored here.
2470 - The " command character is stored here.

2480 - The " command jump address is stored here.
2490 - The # command character is stored here.
2500 - The # command jump address is stored here.
2510 - End of the program.

Listing 10-5

```
4AB2           00100       ORG     4AB2H
4AB2 0000      00110 MEM   DEFW    0
4AB4 21444C    00120 ST    LD      HL,MS1
4AB7 CD752B    00130       CALL    2B75H
4ABA CD4900    00140 L1    CALL    49H
4ABD FE4E      00150       CP      'N'
4ABF CA0043    00160       JP      Z,4300H
4AC2 FE59      00170       CP      'Y'
4AC4 20F4      00180       JR      NZ,L1
4AC6 21BC4C    00190 L2    LD      HL,MS2
4AC9 CD752B    00200       CALL    2B75H
4ACC CD6103    00210       CALL    361H
4ACF 38F5      00220       JR      C,L2
4AD1 D7        00230       RST     16
4AD2 28F2      00240       JR      Z,L2
4AD4 2B        00250       DEC     HL
4AD5 CD0F43    00260       CALL    430FH
4AD8 21B107    00270       LD      HL,1969
4ADB 19        00280       ADD     HL,DE
4ADC 22B24A    00290       LD      (MEM),HL
4ADF 11B14A    00300       LD      DE,4AB1H
4AE2 AF        00310       XOR     A
4AE3 ED52      00320       SBC     HL,DE
4AE5 223E4D    00330       LD      (M1),HL
4AE8 210043    00340       LD      HL,4300H
4AEB 7E        00350 R1    LD      A,(HL)
4AEC FE3F      00360       CP      3FH
4AEE 3832      00370       JR      C,R2
4AF0 FEC2      00380       CP      0C2H
4AF2 D2774B    00390       JP      NC,R3
4AF5 23        00400 R4    INC     HL
4AF6 118545    00410       LD      DE,4585H
4AF9 DF        00420       RST     18H
4AFA CC174C    00430       CALL    Z,IN1
4AFD 11D246    00440       LD      DE,46D2H
4B00 DF        00450       RST     18H
4B01 CC1D4C    00460       CALL    Z,IN2
4B04 117D47    00470       LD      DE,477DH
4B07 DF        00480       RST     18H
4B08 CC214C    00490       CALL    Z,IN3
4B0B 113248    00500       LD      DE,4832H
4B0E DF        00510       RST     18H
4B0F CC254C    00520       CALL    Z,IN4
4B12 112F4A    00530       LD      DE,4A2FH
4B15 DF        00540       RST     18H
4B16 CC294C    00550       CALL    Z,IN5
4B19 116D4A    00560       LD      DE,4A6DH
4B1C DF        00570       RST     18H
4B1D CA2D4C    00580       JP      Z,REND
4B20 18C9      00590       JR      R1
4B22 F5        00600 R2    PUSH    AF
4B23 E630      00610       AND     30H
4B25 B7        00620       OR      A
4B26 2824      00630       JR      Z,ZERO
4B28 FE10      00640       CP      10H
4B2A 2823      00650       JR      Z,TEN
4B2C F1        00660       POP     AF
```

```
4B2D E60F      00670          AND     0FH
4B2F FE02      00680          CP      2
4B31 2824      00690          JR      Z,TR
4B33 FE0A      00700          CP      0AH
4B35 2820      00710          JR      Z,TR
4B37 B7        00720 TEN1     OR      A
4B38 281A      00730          JR      Z.TW
4B3A FE08      00740          CP      8
4B3C 2816      00750          JR      Z,TW
4B3E FE01      00760 ZERO1    CP      1
4B40 2815      00770          JR      Z,TR
4B42 FE06      00780          CP      6
4B44 280E      00790          JR      Z,TW
4B46 FE0E      00800          CP      0EH
4B48 280A      00810          JR      Z,TW
4B4A 18A9      00820          JR      R4
4B4C F1        00830 ZERO     POP     AF
4B4D 18EF      00840          JR      ZERO1
4B4F F1        00850 TEN      POP     AF
4B50 E60F      00860          AND     0FH
4B52 18E3      00870          JR      TEN1
4B54 23        00880 TW       INC     HL
4B55 189E      00890 TW1      JR      R4
4B57 23        00900 TR       INC     HL
4B58 E5        00910          PUSH    HL
4B59 5E        00920          LD      E,(HL)
4B5A 23        00930          INC     HL
4B5B 56        00940          LD      D,(HL)
4B5C EB        00950          EX      DE,HL
4B5D 110043    00960          LD      DE,4300H
4B60 DF        00970          RST     18H
4B61 EB        00980          EX      DE,HL
4B62 380D      00990          JR      C,TRR
4B64 EB        01000          EX      DE,HL
4B65 11B24A    01010          LD      DE,4AB2H
4B68 DF        01020          RST     18H
4B69 EB        01030          EX      DE,HL
4B6A 3005      01040          JR      NC,TRR
4B6C 2A3E4D    01050          LD      HL,(M1)
4B6F 19        01060          ADD     HL,DE
4B70 EB        01070          EX      DE,HL
4B71 E1        01080 TRR      POP     HL
4B72 73        01090          LD      (HL),E
4B73 23        01100          INC     HL
4B74 72        01110          LD      (HL).D
4B75 18DE      01120          JR      TW1
4B77 D6C0      01130 R3       SUB     0C0H
4B79 F5        01140          PUSH    AF
4B7A E630      01150          AND     30H
4B7C B7        01160          OR      A
4B7D 2827      01170          JR      Z,ZER3
4B7F FE10      01180          CP      10H
4B81 2816      01190          JR      Z,ONE
4B83 FE20      01200          CP      20H
4B85 2809      01210          JR      Z,TWO
4B87 F1        01220          POP     AF
4B88 E60F      01230          AND     0FH
4B8A FE0D      01240          CP      0DH
4B8C 283F      01250          JR      Z,DD
4B8E 1823      01260          JR      TRE
4B90 F1        01270 TWO      POP     AF
4B91 E60F      01280          AND     0FH
4B93 FE0D      01290          CP      0DH
4B95 2864      01300          JR      Z,ED
4B97 181A      01310          JR      TRE
4B99 F1        01320 ONE      POP     AF
4B9A E60F      01330          AND     0FH
4B9C FE0D      01340          CP      0DH
```

```
4B9E 282D     01350          JR     Z,DD
4BA0 FE03     01360          CP     3
4BA2 28B0     01370          JR     Z,TW
4BA4 1809     01380          JR     ONE1
4BA6 F1       01390 ZER3     POP    AF
4BA7 FE0D     01400          CP     0DH
4BA9 28AC     01410          JR     Z,TR
4BAB FE03     01420          CP     3
4BAD 28A8     01430          JR     Z,TR
4BAF FE0B     01440 ONE1     CP     0BH
4BB1 28A1     01450          JR     Z,TW
4BB3 FE02     01460 TRE      CP     2
4BB5 28A0     01470          JR     Z,TR
4BB7 FE04     01480          CP     4
4BB9 289C     01490          JR     Z,TR
4BBB FE06     01500          CP     6
4BBD 2895     01510          JR     Z,TW
4BBF FE0A     01520          CP     0AH
4BC1 2894     01530          JR     Z,TR
4BC3 FE0C     01540          CP     0CH
4BC5 2890     01550          JR     Z,TR
4BC7 FE0E     01560          CP     0EH
4BC9 2889     01570          JR     Z,TW
4BCB 1888     01580 TW2      JR     TW1
4BCD 23       01590 DD       INC    HL
4BCE 7E       01600          LD     A,(HL)
4BCF FE21     01610          CP     21H
4BD1 3882     01620          JR     C,TW1
4BD3 2882     01630 TR2      JR     Z,TR
4BD5 FE22     01640          CP     22H
4BD7 28FA     01650          JR     Z,TR2
4BD9 FE2A     01660          CP     2AH
4BDB 28F6     01670          JR     Z,TR2
4BDD FE34     01680          CP     34H
4BDF 2814     01690          JR     Z,DD1
4BE1 FE35     01700          CP     35H
4BE3 2810     01710          JR     Z,DD1
4BE5 FE36     01720          CP     36H
4BE7 280F     01730          JR     Z,DD2
4BE9 FE46     01740          CP     46H
4BEB 38DE     01750          JR     C,TW2
4BED FECB     01760          CP     0CBH
4BEF 3804     01770          JR     C,DD1
4BF1 2805     01780          JR     Z,DD2
4BF3 18D6     01790          JR     TW2
4BF5 23       01800 DD1      INC    HL
4BF6 18D3     01810          JR     TW2
4BF8 23       01820 DD2      INC    HL
4BF9 18FA     01830          JR     DD1
4BFB 23       01840 ED       INC    HL
4BFC 7E       01850          LD     A,(HL)
4BFD FE43     01860          CP     43H
4BFF 28D2     01870          JR     Z,TR2
4C01 FE4B     01880          CP     4BH
4C03 28CE     01890          JR     Z,TR2
4C05 FE53     01900          CP     53H
4C07 28CA     01910          JR     Z,TR2
4C09 FE5B     01920          CP     5BH
4C0B 28C6     01930          JR     Z,TR2
4C0D FE73     01940          CP     73H
4C0F 28C2     01950          JR     Z,TR2
4C11 FE7B     01960          CP     7BH
4C13 28BE     01970          JR     Z,TR2
4C15 18B4     01980          JR     TW2
4C17 068C     01990 IN1      LD     B,140
4C19 23       02000 I1       INC    HL
4C1A 10FD     02010          DJNZ   I1
```

```
4C1C C9         02020          RET
4C1D 061E       02030 IN2      LD      B,30
4C1F 18F8       02040          JR      I1
4C21 0606       02050 IN3      LD      B.6
4C23 18F4       02060          JR      I1
4C25 060B       02070 IN4      LD      B,11
4C27 18F0       02080          JR      I1
4C29 063E       02090 IN5      LD      B,62
4C2B 18EC       02100          JR      I1
4C2D 2B         02110 REND     DEC     HL
4C2E 0617       02120          LD      B,23
4C30 23         02130 RE1      INC     HL
4C31 23         02140          INC     HL
4C32 E5         02150          PUSH    HL
4C33 5E         02160          LD      E,(HL)
4C34 23         02170          INC     HL
4C35 56         02180          LD      D,(HL)
4C36 2A3E4D     02190          LD      HL,(M1)
4C39 19         02200          ADD     HL,DE
4C3A EB         02210          EX      DE,HL
4C3B E1         02220          POP     HL
4C3C 73         02230          LD      (HL),E
4C3D 23         02240          INC     HL
4C3E 72         02250          LD      (HL),D
4C3F 10EF       02260          DJNZ    RE1
4C41 C3F142     02270          JP      42F1H
4C44 1C         02280 MS1      DEFB    1CH
4C45 1F         02290          DEFB    1FH
4C46 53         02300          DEFM    'SWAT V2.0'
4C47 57
4C48 41
4C49 54
4C4A 20
4C4B 56
4C4C 32
4C4D 2E
4C4E 30
4C4F 0D         02310          DEFB    13
4C50 41         02320          DEFM    'A Z-80 SYSTEM MONITOR'
4C51 20
4C52 5A
4C53 2D
4C54 38
4C55 30
4C56 20
4C57 53
4C58 59
4C59 53
4C5A 54
4C5B 45
4C5C 4D
4C5D 20
4C5E 4D
4C5F 4F
4C60 4E
4C61 49
4C62 54
4C63 4F
4C64 52
4C65 0D         02330          DEFB    13
4C66 42         02340          DEFM    'BY MARK D. GOODWIN'
4C67 59
4C68 20
4C69 4D
4C6A 41
4C6B 52
4C6C 4B
4C6D 20
```

```
4C6E 44
4C6F 2E
4C70 20
4C71 47
4C72 4F
4C73 4F
4C74 44
4C75 57
4C76 49
4C77 4E
4C78 0D         02350     DEFB     13
4C79 4D         02360     DEFM     'MONITOR RESIDES AT:
                                   4300H TO 4AB1H'
4C7A 4F
4C7B 4E
4C7C 49
4C7D 54
4C7E 4F
4C7F 52
4C80 20
4C81 52
4C82 45
4C83 53
4C84 49
4C85 44
4C86 45
4C87 53
4C88 20
4C89 41
4C8A 54
4C8B 3A
4C8C 20
4C8D 34
4C8E 33
4C8F 30
4C90 30
4C91 48
4C92 20
4C93 54
4C94 4F
4C95 20
4C96 34
4C97 41
4C98 42
4C99 31
4C9A 48
4C9B 0D         02370     DEFB     13
4C9C 44         02380     DEFM     'DO YOU WISH TO RELOCATE (Y/N)?'
4C9D 4F
4C9E 20
4C9F 59
4CA0 4F
4CA1 55
4CA2 20
4CA3 57
4CA4 49
4CA5 53
4CA6 48
4CA7 20
4CA8 54
4CA9 4F
4CAA 20
4CAB 52
4CAC 45
4CAD 4C
4CAE 4F
4CAF 43
```

```
4CB0 41
4CB1 54
4CB2 45
4CB3 20
4CB4 28
4CB5 59
4CB6 2F
4CB7 4E
4CB8 29
4CB9 3F
4CBA 0D      02390          DEFB     13
4CBB 00      02400          NOP
4CBC 0D      02410 MS2      DEFB     13
4CBD 53      02420          DEFM     'SUGGESTED RELOCATION POINTS:'
4CBE 55
4CBF 47
4CC0 47
4CC1 45
4CC2 53
4CC3 54
4CC4 45
4CC5 44
4CC6 20
4CC7 52
4CC8 45
4CC9 4C
4CCA 4F
4CCB 43
4CCC 41
4CCD 54
4CCE 49
4CCF 4F
4CD0 4E
4CD1 20
4CD2 50
4CD3 4F
4CD4 49
4CD5 4E
4CD6 54
4CD7 53
4CD8 3A
4CD9 0D      02430          DEFB     13
4CDA 54      02440          DEFM     'TOP OF 16K MEMORY - 784EH'
4CDB 4F
4CDC 50
4CDD 20
4CDE 4F
4CDF 46
4CE0 20
4CE1 31
4CE2 36
4CE3 4B
4CE4 20
4CE5 4D
4CE6 45
4CE7 4D
4CE8 4F
4CE9 52
4CEA 59
4CEB 20
4CEC 2D
4CED 20
4CEE 37
4CEF 38
4CF0 34
4CF1 45
4CF2 48
```

```
4CF3 0D         02450         DEFB     13
4CF4 54         02460         DEFM     'TOP OF 32K MEMORY - B84EH'
4CF5 4F
4CF6 50
4CF7 20
4CF8 4F
4CF9 46
4CFA 20
4CFB 33
4CFC 32
4CFD 4B
4CFE 20
4CFF 4D
4D00 45
4D01 4D
4D02 4F
4D03 52
4D04 59
4D05 20
4D06 2D
4D07 20
4D08 42
4D09 38
4D0A 34
4D0B 45
4D0C 48
4D0D 0D         02470         DEFB     13
4D0E 54         02480         DEFM     'TOP OF 48K MEMORY - F84EH'
4D0F 4F
4D10 50
4D11 20
4D12 4F
4D13 46
4D14 20
4D15 34
4D16 38
4D17 4B
4D18 20
4D19 4D
4D1A 45
4D1B 4D
4D1C 4F
4D1D 52
4D1E 59
4D1F 20
4D20 2D
4D21 20
4D22 46
4D23 38
4D24 34
4D25 45
4D26 48
4D27 0D0D       02490         DEFW     0D0DH
4D29 52         02500         DEFM     'RELOCATION ADDRESS: '
4D2A 45
4D2B 4C
4D2C 4F
4D2D 43
4D2E 41
4D2F 54
4D30 49
4D31 4F
4D32 4E
4D33 20
4D34 41
4D35 44
4D36 44
```

```
4D37 52
4D38 45
4D39 53
4D3A 53
4D3B 3A
4D3C 20
4D3D 00        02510          NOP
4D3E 0000      02520 M1       DEFW    0
4AB4           02530          END     ST
00000 TOTAL ERRORS

RE1      4C30
I1       4C19
DD2      4BF8
DD1      4BF5
TR2      4BD3
TW2      4BCB
ONE1     4BAF
ED       4BFB
TRE      4BB3
DD       4BCD
TWO      4B90
ONE      4B99
ZER3     4BA6
TRR      4B71
TW1      4B55
ZERO1    4B3E
TW       4B54
TEN1     4B37
TR       4B57
TEN      4B4F
ZERO     4B4C
REND     4C2D
IN5      4C29
IN4      4C25
IN3      4C21
IN2      4C1D
IN1      4C17
R4       4AF5
R3       4B77
R2       4B22
R1       4AEB
M1       4D3E
MS2      4CBC
L2       4AC6
L1       4ABA
MS1      4C44
ST       4AB4
MEM      4AB2
```

Listing 10-5 Comments

100 - Set the starting address for this part to the end of the monitor program.

110 - The relocation address will be stored here.

120 - Load register pair HL with the starting address of the message to be displayed.

130 - Go display the message.

140 - Go wait till a key is pressed.

150 - Check to see if the key pressed was an N.

160 - Jump if the key pressed was an N.

170 - Check to see if the key pressed was a Y.
180 - Jump if the key pressed wasn't a Y.
190 - Load register pair HL with the starting address of the message to be displayed.
200 - Go display the message.
210 - Go get the keyboard input.
220 - Jump if the BREAK key was pressed.
230 - Bump the input buffer pointer in register pair HL till it points to the first character.
240 - Jump if there wasn't any keyboard input.
250 - Decrement the input buffer pointer in register pair HL.
260 - Go convert the hex input to binary and return with the 16-bit value in register pair DE.
270 - Load register pair HL with the length of the monitor program.
280 - Figure the relocation address by adding the value in register pair DE to the length of the monitor program in register pair HL.
290 - Save the relocation address in register pair HL.
300 - Load register pair DE with the ending address of the monitor program.
310 - Clear the Carry flag
320 - Subtract the ending address of the monitor program in register pair DE from the relocation address in register pair HL.
330 - Save the relocation offset in register pair HL.
340 - Load register pair HL with the starting address of the monitor program.
350 - Load register A with the value at the location of the memory pointer in register pair HL.
360 - Check to see if the value in register A is less than 3FH.
370 - Jump if the value in register A is less than 3FH.
380 - Check to see if the value in register A is greater than or equal to C2H.
390 - Jump if the value in register A is greater than or equal to C2H.
400 - Bump the memory pointer in register pair HL.
410 - Load register pair DE with the location of the first storage area in the monitor program.
420 - Go compare the memory pointer in register pair HL to the location of the first storage area in register pair DE.
430 - Go if the memory pointer in register pair HL matches the

location of the first storage area in register pair DE.

440 - Load register pair DE with the location of the second storage area in the monitor program.

450 - Go compare the memory pointer in register pair HL to the location of the second storage area in register pair DE.

460 - Go if the memory pointer in register pair HL matches the location of the second storage area in register pair DE.

470 - Load register pair DE with the location of the third storage area in the monitor program.

480 - Go compare the memory pointer in register pair HL to the location of the third storage area in register pair DE.

490 - Go if the memory pointer in register pair HL matches the location of the third storage area in register pair DE.

500 - Load register pair DE with the location of the fourth storage area in the monitor program.

510 - Go compare the memory pointer in register pair HL to the location of the fourth storage area in register pair DE.

520 - Go if the memory pointer in register pair HL matches the location of the fourth storage area in register pair DE.

530 - Load register pair DE with the location of the fifth storage area in the monitor program.

540 - Go compare the memory pointer in register pair HL to the location of the fifth storage area in register pair DE.

550 - Go if the memory pointer in register pair HL matches the location of the fifth storage area in register pair DE.

560 - Load register pair DE with the location of the sixth storage area in the monitor program.

570 - Go compare the memory pointer in register pair HL with the location of the sixth storage area in register pair DE.

580 - Go if the memory pointer in register pair HL matches the location of the sixth storage area in register pair DE.

590 - Loop till all of the memory locations have been updated.

600 - Save the value in register A on the stack.

610 - Mask the value in register A.

620 - Check to see if the adjusted value in register A is equal to zero.

630 - Jump if the adjusted value in register A is equal to zero.

640 - Check to see if the adjusted value in register A is equal to 10H.

650 - Jump if the adjusted value in register A is equal to 10H.

660 - Get the value from the stack and put it in register A.

670 - Mask the value in register A.
680 - Check to see if the adjusted value in register A is equal to 02H.
690 - Jump if the adjusted value in register A is equal to 02H.
700 - Check to see if the adjusted value in register A is equal to 0AH.
710 - Jump if the adjusted value in register A is equal to 0AH.
720 - Check to see if the adjusted value in register A is equal to zero.
730 - Jump if the adjusted value in register A is equal to zero.
740 - Check to see if the adjusted value in register A is equal to 08H.
750 - Jump if the adjusted value in register A is equal to 08H.
760 - Check to see if the adjusted value in register A is equal to 01H.
770 - Jump if the adjusted value in register A is equal to 01H.
780 - Check to see if the adjusted value in register A is equal to 06H.
790 - Jump if the adjusted value in register A is equal to 06H.
800 - Check to see if the adjusted value in register A is equal to 0EH.
810 - Jump if the adjusted value in register A is equal to 0EH.
820 - Jump.
830 - Get the value from the stack and put it in register A.
840 - Jump.
850 - Get the value from the stack and put it in register A.
860 - Mask the value in register A.
870 - Jump.
880 - Bump the memory pointer in register pair HL.
890 - Jump.
900 - Bump the memory pointer in register pair HL.
910 - Save the memory pointer in register pair HL on the stack.
920 - Load register E with the LSB of the address at the location of the memory pointer in register pair HL.
930 - Bump the memory pointer in register pair HL.
940 - Load register D with the MSB of the address at the location of the memory pointer in register pair HL.
950 - Load register pair HL with the address in register pair DE.
960 - Load register pair DE with the starting address of the monitor program.
970 - Go check to see if the address in register pair HL is less

than the starting address of the monitor program in register pair DE.

980 - Load register pair DE with the address in register pair HL.

990 - Jump if the address in register pair HL was less than the starting address of the monitor program in register pair DE.

1000 - Load register pair HL with the address in register pair DE.

1010 - Load register pair DE with the ending address of the monitor program.

1020 - Go compare the address in register pair HL with the ending address of the monitor program in register pair DE.

1030 - Load register pair DE with the address in register pair HL.

1040 - Jump if the address in register pair HL was greater than the ending address in register pair DE.

1050 - Load register pair HL with the relocation offset.

1060 - Add the address in register pair DE to the value of the relocation offset in register pair HL.

1070 - Load register pair DE with the updated address.

1080 - Get the memory pointer from the stack and put it in register pair HL.

1090 - Save the LSB of the address in register E at the location of the memory pointer in register pair HL.

1100 - Bump the memory pointer in register pair HL.

1110 - Save the MSB of the address in register D at the location on the memory pointer in register pair HL.

1120 - Jump.

1130 - Adjust the value in register A.

1140 - Save the value in register A on the stack.

1150 - Mask the value in register A.

1160 - Check to see if the value in register A is equal to zero.

1170 - Jump if the value in register A is equal to zero.

1180 - Check to see if the adjusted value in register A is equal to 10H.

1190 - Jump if the adjusted value in register A is equal to 10H.

1200 - Check to see if the adjusted value in register A is equal to 20H.

1210 - Jump if the adjusted value in register A is equal to 20H.

1220 - Get the value from the stack and put it in register A.

1230 - Mask the value in register A.

1240 - Check to see if the adjusted value in register A is equal to 0DH.

1250 - Jump if the adjusted value in register A is equal to 0DH.

1260 - Jump.
1270 - Get the value from the stack and put it in register A.
1280 - Mask the value in register A.
1290 - Check to see if the value in register A is equal to 0DH.
1300 - Jump if the adjusted value in register A is equal to 0DH.
1310 - Jump.
1320 - Get the value from the stack and put it in register A.
1330 - Mask the value in register A.
1340 - Check to see if the value in register A is equal to 0DH.
1350 - Jump if the value in register A is equal to 0DH.
1360 - Check to see if the value in register A is equal to 03H.
1370 - Jump if the adjusted value in register A is equal to 03H.
1380 - Jump.
1390 - Get the value from the stack and put it in register A.
1400 - Check to see if the value in register A is equal to 0DH.
1410 - Jump if the value in register A is equal to 0DH.
1420 - Check to see if the value in register A is equal to 03H.
1430 - Jump if the value in register A is equal to 03H.
1440 - Check to see if the adjusted value in register A is equal to 0BH.
1450 - Jump if the adjusted value in register A is equal to 0BH.
1460 - Check to see if the adjusted value in register A is equal to 02H.
1470 - Jump if the adjusted value in register A is equal to 02H.
1480 - Check to see if the adjusted value in register A is equal to 04H.
1490 - Jump if the adjusted value in register A is equal to 04H.
1500 - Check to see if the adjusted value in register A is equal to 06H.
1510 - Jump if the adjusted value in register A is equal to 06H.
1520 - Check to see if the adjusted value in register A is equal to 0AH.
1530 - Jump if the adjusted value in register A is equal to 0AH.
1540 - Check to see if the adjusted value in register A is equal to 0CH.
1550 - Jump if the adjusted value in register A is equal to 0CH.
1560 - Check to see if the adjusted value in register A is equal to 0EH.
1570 - Jump if the adjusted value in register A is equal to 0EH.
1580 - Jump.
1590 - Bump the memory pointer in register pair HL.

1600 - Load register A with the value at the location of the memory pointer in register pair HL.
1610 - Check to see if the value in register A is less than 21H.
1620 - Jump if the value in register A is less than 21H.
1630 - Jump if the value in register A is equal to 21H.
1640 - Check to see if the value in register A is equal to 22H.
1650 - Jump if the value in register A is equal to 22H.
1660 - Check to see if the value in register A is equal to 2AH.
1670 - Jump if the value in register A is equal to 2AH.
1680 - Check to see if the value in register A is equal to 34H.
1690 - Jump if the value in register A is equal to 34H.
1700 - Check to see if the value in register A is equal to 35H.
1710 - Jump if the value in register A is equal to 35H.
1720 - Check to see if the value in register A is equal to 36H.
1730 - Jump if the value in register A is equal to 36H.
1740 - Check to see if the value in register A is less than 46H.
1750 - Jump if the value in register A is less than 46H.
1760 - Check to see if the value in register A is less than CBH.
1770 - Jump if the value in register A is less than CBH.
1780 - Jump if the value in register A is equal to CBH.
1790 - Jump.
1800 - Bump the memory pointer in register pair HL.
1810 - Jump.
1820 - Bump the memory pointer in register pair HL.
1830 - Jump.
1840 - Bump the memory pointer in register pair HL.
1850 - Load register A with the value at the location of the memory pointer in register pair HL.
1860 - Check to see if the value in register A is equal to 43H.
1870 - Jump if the value in register A is equal to 43H.
1880 - Check to see if the value in register A is equal to 4BH.
1890 - Jump if the value in register A is equal to 4BH.
1900 - Check to see if the value in register A is equal to 53H.
1910 - Jump if the value in register A is equal to 53H.
1920 - Check to see if the value in register A is equal to 5BH.
1930 - Jump if the value in register A is equal to 5BH.
1940 - Check to see if the value in register A is equal to 73H.
1950 - Jump if the value in register A is equal to 73H.
1960 - Check to see if the value in register A is equal to 7BH.
1970 - Jump if the value in register A is equal to 7BH.
1980 - Jump.

1990 - Load register B with the number of bytes to bump the memory pointer in register pair HL.
2000 - Bump the memory pointer in register pair HL.
2010 - Loop till the memory pointer in register pair HL is updated.
2020 - Return.
2030 - Load register B with the number of bytes to bump the memory pointer in register pair HL.
2040 - Jump.
2050 - Load register B with the number of bytes to bump the memory pointer in register pair HL.
2060 - Jump.
2070 - Load register B with the number of bytes to bump the memory pointer in register pair HL.
2080 - Jump.
2090 - Load register B with the number of bytes to bump the memory pointer in register pair HL.
2100 - Jump.
2110 - Decrement the memory pointer in register pair HL.
2120 - Load register B with the number of commands in the monitor's jump table.
2130 - Bump the memory pointer in register pair HL.
2140 - Bump the memory pointer in register pair HL.
2150 - Save the memory pointer in register pair HL on the stack.
2160 - Load register E with the LSB of the address at the location of the memory pointer in register pair HL.
2170 - Bump the memory pointer in register pair HL.
2180 - Load register D with the MSB of the address at the location of the memory pointer in register pair HL.
2190 - Load register pair HL with the relocation offset value.
2200 - Add the address in register pair DE to the offset value in register pair HL.
2210 - Load register pair DE with the updated address in register pair HL.
2220 - Get the memory pointer from the stack and put it in register pair HL.
2230 - Save the LSB of the updated address in register E at the location of the memory pointer in register pair HL.
2240 - Bump the memory pointer in register pair HL.
2250 - Save the MSB of the updated address in register D at the location of the memory pointer in register pair HL.
2260 - Loop till the jump table has been completely updated.

2270 - Jump to the block move routine.
2280 - This will home the cursor as part of the sign-on message.
2290 - This will clear the screen as part of the sign-on message.
2300 - Part of the sign-on message is stored here.
2310 - This will display a carriage return as part of the sign-on message.
2320 - Part of the sign-on message is stored here.
2330 - This will display a carriage return as part of the sign-on message.
2340 - Part of the sign-on message is stored here.
2350 - This will display a carriage return as part of the sign-on message.
2360 - Part of the sign-on message is stored here.
2370 - This will display a carriage return as part of the sign-on message.
2380 - Part of the sign-on message is stored here.
2390 - This will display a carriage return as part of the sign-on message.
2400 - The sign-on message terminator is stored here.
2410 - This will display a carriage return as part of the relocation message.
2420 - Part of the relocation message is stored here.
2430 - This will display a carriage return as part of the relocation message.
2440 - Part of the relocation message is stored here.
2450 - This will display a carriage return as part of the relocation message.
2460 - Part of the relocation message is stored here.
2470 - This will display a carriage return as part of the relocation message.
2480 - Part of the relocation message is stored here.
2490 - This will display two carriage returns as part of the relocation message.
2500 - Part of the relocation message is stored here.
2510 - The relocation message terminator is stored here.
2520 - The relocation offset is stored here.
2530 - End of the program.

Appendix
Level II ROM Memory Map

The Level II ROMs are described location by location in this section. Locations 0043H to 0045H, 0134H, 0137H, 08D8H, 095DH, 0909H, 0AEEH, 0C25H, 0E11H, 0E6BH, 0FA1H, 10F8H, 13E1H, 14B1H, 196BH, 1999H, 199CH, 199FH, 1A17H, 1DBAH, 1E02H, 1E05H, 1E08H, 1F03H to 1F06H, 1F59H, 2682H, 270EH, 2726H, 2799H, 2887H, 2888H, 28C0H, 2A67H, 2D44H, 2D45H, 2D96H, 2E16H, 4018H to 401AH, 4090H to 4092H, 409FH, 412FH, 41E4H to 41E7H are not included here because they are not used or can be ignored.

0000H - 0004H	**POWER UP ROUTINE**
0000H	Turn off the real time clock and the disk interrupts.
0001H	Zero register A.
0002H - 0004H	Go to the Level II BASIC inidalization routine.
0005H - 0007H	Go to RST 0008H code via 4000H.
0008H - 000AH	**RST 0008H**
0008H - 000AH	Go to the RST 0008H vector at 4000H.
000BH - 000CH	**DISK ROUTINE**
000BH	Get the address from the stack and put it in register pair HL.

000CH		Jump to the location of the address in register pair HL.
000DH	- 000FH	**DISK ROUTINE**
000DH	- 000FH	Jump to the disk load and run sector routine.
0010H	- 0012H	**RST 0010H**
0010H	- 0012H	Go to the RST 0010H vector at 4003H.
0013H	- 0017H	**KEYBOARD ROUTINE**
0013H		Save the value in register pair BC on the stack.
0014H	- 0015H	Load register B with the device type entry code.
0016H	- 0017H	Jump to the Level II BASIC driver entry routine.
0018H	- 001AH	**RST 0018H**
0018H	- 001AH	Go to the RST 0018H vector at 4006H.
001BH	- 001FH	**VIDEO AND PRINTER ROUTINE**
001BH		Save the value in register pair BC on the stack.
001CH	- 001DH	Load register B with the device type entry code.
001EH	- 001FH	Jump to the Level II BASIC driver entry routine.
0020H	- 0022H	**RST 0020H**
0020H	- 0022H	Go to the RST 0020H vector at 4009H.
0023H	- 0027H	**DISK ROUTINE**
0023H		Save the value in register pair BC on the stack.
0024H	- 0025H	Load register B with the device type entry code.
0026H	- 0027H	Jump to the Level II BASIC driver entry routine.
0028H	- 002AH	**RST 0028H**
0028H	- 002AH	Go to the RST 0028H vector at 400CH.
002BH	- 002FH	**KEYBOARD ROUTINE**
002BH	- 002DH	Load register pair DE with the starting address of the keyboard device control block.
002EH	- 002FH	Jump to the Level II BASIC driver entry routine.
0030H	- 0032H	**RST 0030H**
0030H	- 0032H	Go to the RST 0030H vector at 400FH.

0033H - 0037H	**VIDEO ROUTINE**	
0033H - 0035H	Load register pair DE with the starting address of the video display device control block.	
0036H - 0037H	Jump to the Level II BASIC driver entry routine.	
0038H - 003AH	**RST 0038H**	
0038H - 003AH	Go to the RST 0038H vector at 4012H.	
003BH - 003FH	**PRINTER ROUTINE**	
003BH - 003DH	Load register pair DE with the starting address of the printer device control block.	
003EH - 003FH	Jump to the Level II BASIC driver entry routine.	
0040H - 0042H	**INPUT ROUTINE**	
0040H - 0042H	Jump to the keyboard input routine.	
0046H - 0048H	**DRIVER ENTRY ROUTINE**	
0046H - 0048H	Jump to the Level II BASIC driver entry routine.	
0049H - 004FH	**KEYBOARD ROUTINE**	
0049H - 004BH	Go scan the keyboard and return with the key pressed, if any, in register A.	
004CH	Check the value in register A to see if a key was pressed.	
004DH	Return if a key was pressed.	
004EH - 004FH	Loop till a key is pressed.	
0050H - 005FH	**KEYBOARD LOOKUP TABLE**	
	Key(s) Pressed	ASCII Value
0050H	ENTER	0DH
0051H	Shift ENTER	0DH
0052H	CLEAR	1FH
0053H	Shift CLEAR	1FH
0054H	BREAK	01H
0055H	Shift BREAK	01H
0056H	Up Arrow	5BH
0057H	Shift Up Arrow	1BH

	Key(s) Pressed	ASCII Value
0058H	Down Arrow	0AH
0059H	Shift Down Arrow	1AH
005AH	Left Arrow	08H
005BH	Shift Left Arrow	18H
005CH	Right Arrow	09H
005DH	Shift Right Arrow	19H
005EH	SPACE	20H
005FH	Shift SPACE	20H

0060H - 0065H **DELAY ROUTINE**

0060H Decrement the counter in register pair BC.

0061H Load register A with the counter's MSB in register B.

0062H Combine the counter's LSB in register C with the counter's MSB in register A.

0063H - 0064H Loop till the counter in register pair BC is equal to zero.

0065H Return.

0066H - 0074H **NMI INTERRUPT ROUTINE** (RESET)

0066H - 0068H Set the stack pointer to 0600H.

0069H - 006BH Load register A with the disk controller status.

006CH Bump the disk controller status in register A.

006DH - 006EH Check the value in register A to see if a disk is present.

006FH - 0071H If a disk is present, go to the Level II BASIC power up routine.

0072H - 0074H Jump to the Level II BASIC READY routine.

0075H - 0104H **INITIALIZATION ROUTINE**

0075H - 0077H Load register pair DE with the ROM storage location of the Level II BASIC division routine.

0078H - 007AH Load register pair HL with the RAM storage location of the Level II BASIC division routine.

007BH - 007DH Load register pair BC with the length of the Level II BASIC division routine.

007EH - 007FH	Move the Level II BASIC division routine to RAM.
0080H - 0082H	Load register pair HL with an RAM memory pointer.
0083H - 0084H	Save a 3AH at the location of the memory pointer in register pair HL.
0085H	Bump the memory pointer in register pair HL.
0086H	Zero the location of the memory pointer in register pair HL.
0087H	Bump the memory pointer in register pair HL.
0088H - 0089H	Save a 2CH at the location of the memory pointer in register pair HL.
008AH	Bump the memory pointer in register pair HL.
008BH - 008DH	Save the value in register pair HL as the starting address of the keyboard input buffer area.
008EH - 0090H	Load register pair DE with the starting address of the Level II BASIC L3 ERROR routine.
0091H - 0092H	Load register B with the number of times to save the jump to the Level II BASIC L3 ERROR routine.
0093H - 0095H	Load register pair HL with the starting address of the Disk Basic links.
0096H - 0097H	Save a C3H at the location of the memory pointer in register pair HL.
0098H	Bump the memory pointer in register pair HL.
0099H	Save the LSB of the Level II BASIC L3 ERROR routine's starting address in register E at the location of the memory pointer in register pair HL.
009AH	Bump the memory pointer in register pair HL.
009BH	Save the MSB of the Level II BASIC L3 ERROR routine's starting address in register D at the location of the memory pointer in register pair HL.
009CH	Bump the memory pointer in register pair HL.
009DH - 009EH	Loop till all of the Disk Basic links have been set to jump to the Level II BASIC L3 ERROR routine.
009FH - 00A0H	Load register B with the number of DOS links to set to RETs.
00A1H - 00A2H	Save a C9H at the location of the memory pointer in register pair HL.

00A3H	Bump the memory pointer in register pair HL.
00A4H	Bump the memory pointer in register pair HL.
00A5H	Bump the memory pointer in register pair HL.
00A6H - 00A7H	Loop till all of the DOS links have been set to RETs.
00A8H - 00AAH	Load register pair HL with the starting address of user RAM.
00ABH	Zero the location of the memory pointer in register pair HL.
00ACH - 00AEH	Set the current stack pointer to 41F8H.
00AFH - 00B1H	Go initialize the Level II BASIC variables and pointers.
00B2H - 00B4H	Go clear the screen.
00B5H - 00B7H	Load register pair HL with the starting address of the memory size? message.
00B8H - 00BAH	Go display the memory size? message.
00BBH - 00BDH	Go get the input from the keyboard.
00BEH - 00BFH	Go ask again if the BREAK key was pressed.
00C0H	Bump the input buffer pointer in register pair HL till it points to the first character of the input.
00C1H	Check to see if the character at the location of the input buffer pointer in register A is an end of the input character (00H).
00C2H - 00C3H	Jump if there was a response to the memory size? question.
00C4H - 00C6H	Load register pair HL with the starting address for the memory size check.
00C7H	Bump the memory pointer in register pair HL.
00C8H	Load register A with the MSB of the current memory pointer in register H.
00C9H	Combine the LSB of the current memory pointer in register L with the MSB of the current memory pointer in register A.
00CAH - 00CBH	Jump if the current memory pointer in register pair HL is equal to zero.
00CCH	Load register A with the value at the location of the current memory pointer in register pair HL.

00CDH	Load register B with the value in register A.
00CEH	Complement the value in register A.
00CFH	Save the value in register A at the location of the current memory pointer in register pair HL.
00D0H	Check to see if the value at the location of the memory pointer in register pair HL is the same as the value in register A.
00D1H	Save the value in register B at the location of the memory pointer in register pair HL.
00D2H - 00D3H	Loop till the end of memory is found.
00D4H - 00D5H	Jump.
00D6H - 00D8H	Go evaluate the response to the memory size? question and return with the memory size? in register pair De.
00D9H	Check to see if register A is equal to zero.
00DAH - 00DCH	Go to the Level II BASIC error routine and display a SN ERROR message if register A is equal to zero.
00DDH	Load register pair HL with the memory size? in register pair DE.
00DEH	Decrement the memory size? in register pair HL.
00DFH - 00E0H	Load register A with a memory test value.
00E1H	Load register B with the value at the location of the memory size? pointer in register pair HL.
00E2H	Save the value in register A at the location of the memory size? pointer in register pair HL.
00E3H	Check to see if the value at the location of the memory size? pointer in register pair HL is the same as the value in register A.
00E4H	Save the value in register B at the location of the memory size? pointer in register pair HL.
00E5H - 00E6H	Jump if the memory location doesn't exist.
00E7H	Decrement the memory size pointer in register pair HL.
00E8H - 00EAH	Load register pair DE with the minimum memory size? response.
00EBH	Go check to see if the memory size? pointer in

register pair HL is less than the minimum memory size? response in register pair DE.

00ECH - 00EEH	Go to the Level II BASIC error routine and display an OM ERROR if the memory size? pointer in register pair HL is less than the minimum memory size? response in register pair DE.
00EFH - 00F1H	Load register pair DE with a negative fifty.
00F2H - 00F4H	Save the memory size? pointer in register pair HL.
00F5H	Add the value in register pair DE to the memory size? pointer in register pair HL.
00F6H - 00F8H	Save the start of string space pointer in register pair HL.
00F9H - 00FBH	Go reset the Level II BASIC variables and pointers.
00FCH - 00FEH	Load register pair HL with the starting address of the RADIO SHACK LEVEL II BASIC message.
00FFH - 0101H	Go display the RADIO SHACK LEVEL II BASIC message.
0102H - 0104H	Go to the Level II BASIC READY routine.
0105H - 0110H	**MESSAGE STORAGE**
0105H - 0110H	Memory size? message is stored here.
0111H - 012CH	**MESSAGE STORAGE**
0111H - 012CH	RADIO SHACK LEVEL II BASIC message is stored here.
012DH - 0131H	**L3 ERROR ROUTINE**
012DH - 012EH	Load register E with the L3 ERROR code.
012FH - 0131H	Go to the Level II BASIC error routine.
0132H - 0134H	**LEVEL II BASIC POINT COMMAND ENTRY POINT**
0132H	Go bump the current BASIC program pointer till it points to the next character.
0133H	Zero register A.
0135H - 0137H	**LEVEL II BASIC SET COMMAND ENTRY POINT**
0135H - 0136H	Load register A with 80H.

0138H - 0139H **LEVEL II BASIC RESET COMMAND ENTRY POINT**

0138H - 0139H Load register A with 01H.

013AH - 019CH **GRAPHICS ROUTINE**

013AH Save the graphics mode in register A on the stack.

013BH - 013CH Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a (character.

013DH - 013FH Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the 8-bit value in register A.

0140H - 0141H Check to see if the X value in register A is greater than 127.

0142H - 0144H Go to the Level II BASIC error routine and display a FC ERROR message if the X value in register A is greater than 127.

0145H Save the X value in register A on the stack.

0146H - 0147H Check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a comma.

0148H - 014AH Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the 8-bit value in register A.

014BH - 014CH Check to see if the Y value in register A is greater than 47.

014DH - 014FH Go to the Level II BASIC error routine and display a FC ERROR message if the Y value in register A is greater than 47.

0150H - 0151H Load register D with starting quotient.

0152H Bump the quotient in register D.

0153H - 0154H Subtract 3 from register A.

0155H - 0156H Loop till register D equals the Y value divided by 3.

0157H - 0158H Adjust the remainder in register A.

0159H Save the remainder in register C.

015AH Get the X value from the stack and put it in register A.

015BH Multiply the X value in register A by two.

015CH	Load register E with the X value times two in register A.
015DH - 015EH	Load register B with the number of times to shift register pair DE.
015FH	Load register A with the adjusted Y value in register D.
0160H	Divide the adjusted Y value in register A by two.
0161H	Save the new Y value in register A in register D.
0162H	Load register A with the adjusted X value in register E.
0163H	Divide the adjusted X value in register A by two.
0164H	Load register E with the new X value in register A.
0165H - 0166H	Loop till the memory offset in register pair DE has been figured.
0167H	Load register A with the value in register C.
0168H	Multiply the value in register A by two and add the value of the Carry flag to register A.
0169H	Bump the value in register A.
016AH	Save the bit position in register A in register B.
016BH	Zero register A and reset the Carry flag.
016CH	Set the Carry flag.
016DH	Add the value of the Carry flag to register A.
016EH - 016FH	Loop till the graphic mask has been completed in register A.
0170H	Save the graphic mask in register A in register C.
0171H	Load register A with the MSB of the video memory offset in register D.
0172H - 0173H	Mask the MSB of the video memory offset in register A so that it will point to the correct location in video memory.
0174H	Save the MSB of the video memory pointer in register A in register D.
0175H	Load register A with the character at the location of the video memory pointer in register pair DE.

0176H	Check to see if the character in register A is a graphic character.
0177H - 0179H	Jump if the character in register A is a graphic character.
017AH - 017BH	Load register A with a blank graphic character.
017CH	Save the graphic character in register A in register B.
017DH	Get the graphic mode from the stack and put it in register A.
017EH	Set the flags according to the graphic mode in register A.
017FH	Get the graphic character from register B and put it in register A.
0180H - 0181H	Jump if the graphic mode is POINT.
0182H	Save the graphic character in register A at the location of the video memory pointer in register pair DE.
0183H - 0185H	Jump if the graphic mode is SET.
0186H	Load register A with the graphic mask in register C.
0187H	Reverse the graphic mask in register A.
0188H	Load register C with the adjusted graphic mask in register A.
0189H	Load register A with the character at the location of the video memory pointer in register pair DE.
018AH	RESET the graphic bit by combining the graphic mask in register C with the graphic character in register A.
018BH	Save the adjusted graphic character in register A at the location of the video memory pointer in register pair DE.
018CH - 018DH	Check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a) character.
018EH	Return.
018FH	SET the graphic bit by combining the graphic mask in register C with the graphic character in register A.

0190H - 0191H		Jump.
0192H		POINT the graphic bit by combining the graphic mask in register C with the graphic character in register A.
0193H - 0194H		Subtract one from the value in register A.
0195H		Adjust the value in register A so that A will equal zero if the bit was off in register A.
0196H		Save the current BASIC program pointer in register pair HL on the stack.
0197H - 0199H		Save the value in register A as the current result in REG1.
019AH		Get the current BASIC program pointer from the stack and put it in register pair HL.
019BH - 019CH		Jump.
019DH - 01C8H		**LEVEL II BASIC INKEY$ ROUTINE**
019DH		Go bump the current BASIC program pointer in register pair HL till it points to the next character.
019EH		Save the current BASIC program pointer in register pair HL on the stack.
019FH - 01A1H		Load register A with the last key pressed.
01A2H		Check to see if a key has been pressed.
01A3H - 01A4H		Jump if a key has been pressed.
01A5H - 01A7H		Go scan the keyboard.
01A8H		Check to see if a key was pressed.
01A9H - 01AAH		Jump if a key wasn't pressed.
01ABH		Save the key pressed in register A on the stack.
01ACH		Zero register A.
01ADH - 01AFH		Save the value in register A as the last key pressed.
01B0H		Bump the value in register A.
01B1H - 01B3H		Go make an entry in string space.
01B4H		Get the last key pressed from the stack and put it in register A.
01B5H - 01B7H		Load register pair HL with the string's starting address.

01B8H	Save the last key pressed in register A at the location of the string pointer in register pair HL.
01B9H - 01BBH	Go save the string's VARPTR as the current result.
01BCH - 01BEH	Load register pair HL with the starting address of the READY message.
01BFH - 01C1H	Save the address in register pair HL as the current result in REG1.
01C2H - 01C3H	Load register A with a string number type flag.
01C4H - 01C6H	Save the value in register A as the current number type.
01C7H	Get the current BASIC program pointer from the stack and put it in register pair HL.
01C8H	Return.
01C9H - 01D2H	**LEVEL II BASIC CLS ROUTINE**
01C9H - 01CAH	Load register A with the ASCII character to home the cursor.
01CBH - 01CDH	Go send the character in register A to the video display.
01CEH - 01CFH	Load register A with the ASCII character to clear to the end of the screen.
01D0H - 01D2H	Go send the character in register A to the video display.
01D3H - 01D8H	**LEVEL II BASIC RANDOM ROUTINE**
01D3H - 01D4H	Load register A with the current value of the refresh register.
01D5H - 01D7H	Save the value in register A as part of the random number seed.
01D8H	Return.
01D9H - 01F7H	**CASSETTE ROUTINE**
01D9H - 01DBH	Load register pair HL with the command to send to the cassette.
01DCH - 01DEH	Go send the command to the cassette controller.
01DFH - 01E0H	Load register B with the delay count.
01E1H - 01E2H	Loop for delay.

01E3H - 01E5H	Load register pair HL with the command to send to the cassette.
01E6H - 01E8H	Go send the command to the cassette controller.
01E9H - 01EAH	Load register B with the delay count.
01EBH - 01ECH	Loop for delay.
01EDH - 01EFH	Load register pair HL with the command to send to the cassette.
01F0H - 01F2H	Go send the command to the cassette controller.
01F3H - 01F4H	Load register B with the delay count.
01F5H - 01F6H	Loop for delay.
01F7H	Return.
01F8H - 01FDH	**CASSETTE ROUTINE** (TURN OFF CASSETTE)
01F8H	Save the value in register pair HL on the stack.
01F9H - 01FBH	Load register pair HL with the command to send to the cassette.
01FCH - 01FDH	Jump.
01FEH - 0211H	**CASSETTE ROUTINE** (EVALUATE DRIVE NUMBER)
01FEH	Get the character at the location of the current BASIC program pointer and put it in register A.
01FFH - 0200H	Check to see if the character in register A is a # character.
0201H - 0202H	Zero register A.
0203H - 0204H	Jump if the character at the location of the current BASIC program pointer in register pair HL isn't a # character.
0205H - 0207H	Go evaluate the drive number at the location of the current BASIC program pointer in register pair HL and return with the drive number in register E.
0208H - 0209H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a comma.
020AH	Load register A with the drive number in register E.
020BH	Combine the MSB of the drive number in register D with the LSB of the drive number in register A.

020CH - 020DH	Make the drive number in register A positive.
020EH - 0210H	Go to the Level II BASIC error routine and display a FC ERROR message if the drive number in register A is invalid.
0211H	Decrement the drive number in register A.
0212H - 021DH	**CASSETTE ROUTINE** (TURN ON CASSETTE)
0212H - 0214H	Set the current drive as specified by register A.
0215H	Save the current BASIC program pointer in register pair HL on the stack.
0216H - 0218H	Load register pair HL with the command to send to the cassette.
0219H - 021BH	Go send the command to the cassette controller.
021CH	Get the current BASIC program pointer from the stack and put it in register pair HL.
021DH	Return.
021EH - 022BH	**CASSETTE ROUTINE**
021EH - 0220H	Load register pair HL with the command to send to the cassette.
0221H - 0223H	Load register A with the 32/64 character per line flag.
0224H	Combine the value in register H with the 32/64 character per line flag in register A.
0225H	Combine the value to send to the cassette in register L with the adjusted value in register A.
0226H - 0227H	Send the value in register A to the cassette port.
0228H - 022AH	Save the value in register A as the 32/64 character per line flag.
022BH	Return.
022CH - 0234H	**CASSETTE ROUTINE** (BLINK **)
022CH - 022EH	Get the character being displayed in the upper right hand corner of the video display and put it in register A.
022FH - 0230H	If the character in register A is a * then make it a space, else if the character in register A is a space make it a *.
0231H - 0233H	Display the character in register A in the upper right

	hand corner of the video display.
0234H	Return.
0235H - 0240H	**CASSETTE ROUTINE** (READ A BYTE)
0235H	Save the value in register pair BC on the stack.
0236H	Save the value in register pair HL on the stack.
0237H - 0238H	Load register B with the number of bits to read.
0239H - 023BH	Go read a bit from the cassette recorder.
023CH - 023DH	Loop till all eight bits have been read.
023EH	Get the value from the stack and put it in register pair HL.
023FH	Get the value from the stack and put it in register pair BC.
0240H	Return.
0241H - 0260H	**CASSETTE ROUTINE** (READ A BIT)
0241H	Save the value in register pair BC on the stack.
0242H	Save the value in register A on the stack.
0243H - 0244H	Read a bit from the cassette port.
0245H	Check to see if this is a start bit.
0246H - 0247H	Loop till start bit is found.
0248H - 0249H	Load register B with the delay count.
024AH - 024BH	Loop for delay.
024CH - 024EH	Go clear the cassette controller.
024FH - 0250H	Load register B with the delay count.
0251H - 0252H	Loop for delay.
0253H - 0254H	Load register A with the value of the cassette port.
0255H	Load register B with the value read from the cassette port in register A.
0256H	Get the value from the stack and put it in register A.
0257H - 0258H	Shift the bit read in register B into the Carry flag.
0259H	Shift the value in the Carry flag into register A.

025AH		Save the value in register A on the stack.
025BH	- 025DH	Go clear the cassette controller.
025EH		Get the value from the stack and put it in register A.
025FH		Get the value from the stack and put it in register pair BC.
0260H		Return.
0261H	- 0263H	**CASSETTE ROUTINE**
0261H	- 0263H	Go write a byte.
0264H	- 027DH	**CASSETTE ROUTINE** (WRITE A BYTE)
0264H		Save the value in register pair HL on the stack.
0265H		Save the value in register pair BC on the stack.
0266H		Save the value in register pair DE on the stack.
0267H		Save the value in register A on the stack.
0268H	- 0269H	Load register C with the number of bits to be written.
026AH		Load register D with the byte to be written in register A.
026BH	- 026DH	Go write the start bit.
026EH		Load register A with the byte to be written in register D.
026FH		Shift the bit to be written into the Carry flag.
0270H		Save the adjusted byte to be written in register A in register D.
0271H	- 0272H	Jump if the bit to be written is equal to zero.
0273H	- 0275H	Go make a cassette pulse.
0276H		Decrement the counter in register C.
0277H	- 0278H	Loop till all eight bits have been written.
0279H		Get the value from the stack and put it in register A.
027AH		Get the value from the stack and put it in register pair DE.
027BH		Get the value from the stack and put it in register pair BC.

027CH		Get the value from the stack and put it in register pair HL.
027DH		Return.
027EH	- 0283H	**CASSETTE ROUTINE**
027EH	- 027FH	Load register B with the delay count.
0280H	- 0281H	Loop for the delay.
0282H	- 0283H	Jump.
0284H	- 0292H	**CASSETTE ROUTINE** (TURN ON CASSETTE AND WRITE LEADER)
0284H	- 0286H	Go evaluate the drive number and turn on the cassette motor.
0287H	- 0288H	Load register B with the number of bytes to be written.
0289H		Zero register A.
028AH	- 028CH	Go write the byte in register A.
028DH	- 028EH	Loop till leader has been written.
028FH	- 0290H	Load register A with the sync byte value.
0291H	- 0292H	Go write the sync byte.
0293H	- 029EH	**CASSETTE ROUTINE** (TURN ON CASSETTE AND READ LEADER)
0293H	- 0295H	Go evaluate the drive number and turn on the cassette motor.
0296H		Save the current BASIC program pointer in register pair HL on the stack.
0297H		Zero register A.
0298H	- 029AH	Go read a byte from the cassette and return with it in register A.
029BH	- 029CH	Check to see if the byte read from the cassette in register A is a sync byte.
029DH	- 029EH	Loop till sync byte found.
029FH	- 02A8H	**CASSETTE ROUTINE** (DISPLAY ** ON VIDEO)
029FH	- 02A0H	Load register A with a * character.
02A1H	- 02A3H	Display the * character in register A on the video display.

02A4H - 02A6H		Display the * character in register A on the video display.
02A7H		Get the current BASIC program pointer from the stack and put it in register pair HL.
02A8H		Return.
02A9H - 0329H		**LEVEL II SYSTEM ROUTINE**-ENTRY POINT - 02B2H
02A9H - 02ABH		Go read the execution address from the cassette and return with it in register pair HL.
02ACH - 02AEH		Save the execution address in register pair HL.
02AFH - 02B1H		Go turn off the cassette motor.
02B2H - 02B4H		Go call the DOS link at 41E2H.
02B5H - 02B7H		Set the stack pointer to 4288H.
02B8H - 02BAH		Go display a carriage return on the video display if necessary.
02BBH - 02BCH		Load register A with an * character.
02BDH - 02BFH		Go display the * character in register A on the video display.
02C0H - 02C2H		Go get the input from the keyboard.
02C3H - 02C5H		Go to the Level II BASIC READY routine if the BREAK key was pressed.
02C6H		Go bump the input buffer pointer in register pair HL till it points to the first character input.
02C7H - 02C9H		Go to the Level II BASIC error routine and display a SN ERROR message if there wasn't any input.
02CAH - 02CBH		Check to see if the character at the location of the input buffer pointer in register A is a / character.
02CCH - 02CDH		Jump if the character at the location of the input buffer pointer in register A is a / character.
02CEH - 02D0H		Go turn on the cassette motor.
02D1H - 02D3H		Go read a byte from the cassette and return with it in register A.
02D4H - 02D5H		Check to see if the byte read from the cassette in register A is a header byte.
02D6H - 02D7H		Loop till the header byte is found.

02D8H - 02D9H	Load register B with the length of the filename to read from the cassette.
02DAH	Load register A with the character at the location of the current input buffer pointer in register pair HL.
02DBH	Check to see if the character at the location of the current input buffer pointer in register A is an end of input character.
02DCH - 02DDH	Jump if the character at the location of the current input buffer pointer in register A is an end of input character.
02DEH - 02E0H	Go read a byte from the cassette and return with it in register A.
02E1H	Check to see if the character at the location of the current input buffer pointer in register pair HL is the same as the character read from the cassette in register A.
02E2H - 02E3H	Jump if the character at the location of the current input buffer pointer in register pair HL isn't the same as the character read from the cassette in register A.
02E4H	Bump the input buffer pointer in register pair HL.
02E5H - 02E6H	Loop till the whole of the filename has been read from the cassette and checked against the user response.
02E7H - 02E9H	Go blink the * on the video display.
02EAH - 02ECH	Go read a byte from the cassette recorder and return with it in register A.
02EDH - 02EEH	Check to see if the byte read from the cassette in register A is an execution address header byte.
02EFH - 02F0H	Jump if the byte read from the cassette in register A is an execution address header byte.
02F1H - 02F2H	Check to see if the byte read from the cassette in register A is a file block header byte.
02F3H - 02F4H	Loop till either an execution address header byte or a file block header byte is read from the cassette.
02F5H - 02F7H	Go read a byte from the cassette and return with the number of bytes to be loaded in register B.
02F8H	Load register B with the number of bytes to be loaded in register A.
02F9H - 02FBH	Go read the file block's starting address from the

	cassette and return with it in register pair HL.
02FCH	Add the LSB of the file block's starting address in register L to the MSB of the file block's starting address in register A.
02FDH	Load register C with the file block's starting checksum in register A.
02FEH - 0300H	Go read a byte from the cassette and return with it in register A.
0301H	Save the byte read from the cassette in register A at the location of the memory pointer in register pair HL.
0302H	Bump the memory pointer in register pair HL.
0303H	Add the value of the current checksum in register C to the value in register A.
0304H	Load register C with the updated checksum in register A.
0305H - 0306H	Loop till the whole file block has been read.
0307H - 0309H	Go read the checksum from the cassette and return with it in register A.
030AH	Check to see if the computed checksum in register C is the same as the checksum read from the cassette in register A.
030BH - 030CH	Jump if the computed checksum in register C is the same as the checksum read from the cassette in register A.
030DH - 030EH	Load register A with a C character.
030FH - 0311H	Display the C character in register A on the video display.
0312H - 0313H	Jump.
0314H - 0316H	Go read a byte from the cassette and return with it in register A.
0317H	Load register L with the byte read from the cassette in register A.
0318H - 031AH	Go read a byte from the cassette and return with it in register A.
031BH	Load register H with the byte read from the cassette in register A.
031CH	Return.

031DH	Load register pair DE with the input buffer pointer in register pair HL.
031EH - 0320H	Load register pair HL with the current execution address.
0321H	Exchange the execution address in register pair HL with the input buffer pointer in register pair DE.
0322H	Go bump the current input buffer pointer in register pair HL till it points to the next character.
0323H - 0325H	Go evaluate the expression at the location of the input buffer pointer in register pair HL and return with the integer result in register pair DE.
0326H - 0327H	Jump if there wasn't any input.
0328H	Exchange the input buffer pointer in register pair HL with the execution address in register pair DE.
0329H	Jump to the execution address in register pair HL.
032AH - 0347H	**OUTPUT ROUTINE**
032AH	Save the value in register pair BC on the stack.
032BH	Load register C with the character to be output in register A.
032CH - 032EH	Go call the DOS link at 41C1H.
032FH - 0331H	Load register A with the current output device number.
0332H	Set the flags according to the current output device number in register A.
0333H	Load register A with the character to be output in register C.
0334H	Get the value from the stack and put it in register pair BC.
0335H - 0337H	Jump if the character in register A is to be sent to the cassette.
0338H - 0339H	Jump if the character in register A is to be sent to the printer.
033AH	Save the value in register pair DE on the stack.
033BH - 033DH	Go send the character in register A to the video display.
033EH	Save the character in register A on the stack.

033FH - 0341H		Go update the current cursor position.
0342H - 0344H		Save the current cursor line position in register A.
0345H		Get the character from the stack and put it in register A.
0346H		Get the value from the stack and put it in register pair DE.
0347H		Return.
0348H - 0357H		**VIDEO ROUTINE**
0348H - 034AH		Load register A with the 32/64 character per line flag.
034BH - 034CH		Set the flags according to the status of the 32/64 character per line flag in register A.
034DH - 034FH		Load register A with the LSB of the current cursor position.
0350H - 0351H		Jump if this is the 64 character per line mode.
0352H		Divide the LSB of the current cursor position in register A by two.
0353H - 0354H		Mask the cursor line position in register A for a 32 character per line.
0355H - 0356H		Mask the cursor line position in register A for 64 characters per line.
0357H		Return.
0358H - 0360H		**KEYBOARD ROUTINE**
0358H - 035AH		Go call the DOS link at 41C4H.
035BH		Save the value in register pair DE on the stack.
035CH - 035EH		Go scan the keyboard.
035FH		Get the value from the stack and put it in register pair DE.
0360H		Return.
0361H - 0383H		**INPUT ROUTINE**
0361H		Zero register A.
0362H - 0364H		Save the value in register A as the last key pressed.
0365H - 0367H		Save the value in register A as the current cursor line position.

0368H - 036AH	Go call the DOS link at 41AFH.
036BH	Save register pair BC on the stack.
036CH - 036EH	Load register pair HL with the starting address of the input buffer.
036FH - 0370H	Load register B with the length of the input buffer.
0371H - 0373H	Go get the input.
0374H	Save the flags on the stack.
0375H	Load register C with the length of the input in register B.
0376H - 0377H	Zero register B.
0378H	Add the length of the input in register pair BC to the starting address of the input buffer in register pair HL.
0379H - 037AH	Save an end of the input character at the location of the end of input pointer in register pair HL.
037BH - 037DH	Load register pair HL with the starting address of the input buffer.
037EH	Get the flags from the stack.
037FH	Get the value from the stack and put it in register pair BC.
0380H	Decrement the input buffer pointer in register pair HL.
0381H	Return if the BREAK key was pressed.
0382H	Zero register A.
0383H	Return.
0384H - 038AH	**KEYBOARD ROUTINE**
0384H - 0386H	Go scan the keyboard.
0387H	Check to see if a key was pressed.
0388H	Return if a key was pressed.
0389H - 038AH	Loop till a key is pressed.
038BH - 039BH	**PRINTER ROUTINE**
038BH	Zero register A.

038CH - 038EH	Save the value in register A as the current output device code.
038FH - 0391H	Load register A with the current printer carriage position.
0392H	Check to see if the carriage position in register A is equal to zero.
0393H	Return if the carriage position in register A is equal to zero.
0394H - 0395H	Load register A with a carriage return character.
0396H	Save the value in register pair DE on the stack.
0397H - 0399H	Go send the carriage return character in register A to the printer.
039AH	Get the value from the stack and put it in register pair DE.
039BH	Return.
039CH - 03C1H	**PRINTER ROUTINE**
039CH	Save the value in register pair AF on the stack.
039DH	Save the value in register pair DE on the stack.
039EH	Save the value in register pair BC on the stack.
039FH	Load register C with the character to be sent to the printer in register A.
03A0H - 03A1H	Zero register E.
03A2H - 03A3H	Check to see if the character to be sent to the printer in register A is equal to 0CH.
03A4H - 03A5H	Jump if the character to be sent to the printer in register A is equal to 0CH.
03A6H - 03A7H	Check to see if the character to be sent to the printer in register A is a line feed character.
03A8H - 03A9H	Jump if the character to be sent to the printer in register A isn't a line feed character.
03AAH - 03ABH	Load register A with a carriage return character.
03ACH	Load register C with the character to be sent to printer in register A.
03ADH - 03AEH	Check to see if the character to be sent to the printer in register A is a carriage return character.

03AFH - 03B0H	Jump if the character to be sent to the printer in register A is a carriage return character.
03B1H - 03B3H	Load register A with the current carriage position.
03B4H	Bump the current carriage position in register A.
03B5H	Load register E with the current carriage position in register A.
03B6H	Load register A with the current carriage position in register E.
03B7H - 03B9H	Save the current carriage position in register A.
03BAH	Load register A with the character to be sent to the printer in register C.
03BBH - 03BDH	Go send the character in register A to the printer.
03BEH	Get the value from the stack and put it in register pair BC.
03BFH	Get the value from the stack and put it in register pair DE.
3C0H	Get the value from the stack and put it in register pair AF.
03C1H	Return.
03C2H - 03E2H	**DRIVER ENTRY ROUTINE**
03C2H	Save register pair HL on the stack.
03C3H - 03C4H	Save the value in register pair IX on the stack.
03C5H	Save the starting address of the device control block in register pair DE on the stack.
03C6H - 03C7H	Get the starting address of the device control block from the stack and put it in register pair IX.
03C8H	Save the value in register pair DE on the stack.
03C9H - 03CBH	Load register pair HL with the return address.
03CCH	Save the return address in register pair HL on the stack.
03CDH	Save the character to output in register A in register C.
03CEH	Load register A with the device type code at the location of the device control block pointer in register pair DE.

03CFH	Mask the device type code in register A with the driver entry code in register B.
03D0H	Check to see if the updated device type code in register A is the same as the driver entry code in register B.
03D1H - 03D3H	Jump to the DOS link at 4033H if the updated device type code in register A isn't the same as the driver entry code in register B.
03D4H - 03D5H	Reset the flags.
03D6H - 03D8H	Load register L with the LSB of the driver entry address at the location of the device control block pointer in register pair IX plus one.
03D9H - 03DBH	Load register H with the MSB of the driver entry address at the location of the device control block pointer in register pair IX plus one.
03DCH	Jump to the driver entry address in register pair HL.
03DDH	Get the value from the stack and put it in register pair DE.
03DEH - 03DFH	Get the value from the stack and put it in register pair IX.
03E0H	Get the value from the stack and put it in register pair HL.
03E1H	Get the value from the stack and put it in register pair BC.
03E2H	Return.
03E3H - 0457H	**KEYBOARD DRIVER**
03E3H - 03E5H	Load register pair HL with the keyboard work area's starting address.
03E6H - 03E8H	Load register pair BC with the keyboard memory's starting address.
03E9H - 03EAH	Zero register D.
03EBH	Load register A with the value at the location of the keyboard memory pointer in register pair BC.
03ECH	Load register E with the keyboard memory value in register A.
03EDH	Check for inequality by XORing the value at the location of the keyboard work area pointer in register pair HL with the keyboard memory value in register A.

03EEH	Save the keyboard memory value in register E at the location of the keyboard work area pointer in register pair HL.
3EFH	Mask the adjusted value in register A with the value at the location of the keyboard work area pointer in register pair HL.
03F0H - 03F1H	Jump if a new key has been pressed.
03F2H	Bump the keyboard row counter in register D.
03F3H	Bump the keyboard work area pointer in register pair HL.
03F4H - 03F5H	Adjust the keyboard memory pointer in register pair BC.
03F6H - 03F8H	Jump if the whole of keyboard memory hasn't been checked.
03F9H	Return.
03FAH	Save the column number in register A in register E.
03FBH	Load register A with the row counter in register D.
03FCH	Multiply the row counter in register A by two.
03FDH	Multiply the row counter in register A by two.
03FEH	Multiply the row counter in register A by two.
03FFH	Load register D with the row counter in register A.
0400H - 0401H	Load register C with the starting column counter.
0402H	Load register A with the column counter in register C.
0403H	Check to see if the column counter in register A is the same as the active column number in register E.
0404H - 0405H	Jump if the column counter in register A is the same as the active column number in register E.
0406H	Bump the value in register D.
0407H - 0408H	Adjust the column counter in register C.
0409H - 040AH	Loop till the value in register D is adjusted for it's column.
040BH - 040DH	Load register A with the value of the SHIFT memory location.

040EH	Load register B with the SHIFT value in register A.
040FH	Load register A with the value in register D.
0410H - 0411H	Adjust the ASCII value in register A.
0412H - 0413H	Check to see if the value in register A is alphabetic.
0414H - 0415H	Jump if the value in register A is nonalphabetic.
0416H - 0417H	Put the SHIFT value in register B in the carry flag.
0418H - 0419H	Jump if the SHIFT key wasn't pressed.
041AH - 041BH	Adjust the value in register A for lower case.
041CH	Load register D with the adjusted character in register A.
041DH - 041FH	Load register A with the value at keyboard memory row six.
0420H - 0421H	Check to see if the down arrow key was pressed.
0422H - 0423H	Jump if down arrow wasn't pressed.
0424H	Load register A with the character in register D.
0425H - 0426H	Adjust the value of the character in register A.
0427H - 0428H	Jump.
0429H - 042AH	Adjust the value of the character in register A.
042BH - 042CH	Jump if the character in register A is for keyboard row six.
042DH - 042EH	Readjust the value of character in register A.
042FH - 0430H	Check to see if the character in register A is 0 - , character.
0431H - 0432H	Jump if the character in register A is a 0 - , character.
0433H - 0434H	Adjust the character in register A.
0435H - 0436H	Put the SHIFT value in register B into the Carry flag.
0437H - 0438H	Jump if the SHIFT key wasn't pressed.
0439H - 043AH	Adjust the character in register A.
043BH - 043CH	Jump.

043DH	Adjust the value in register A.
043EH - 043FH	Put the SHIFT value in register B into the Carry flag.
0440H - 0441H	Jump if the SHIFT key wasn't pressed.
0442H	Bump the value in register A.
0443H - 0445H	Load register pair HL with the starting address of the keyboard lookup table.
0446H	Load register C with the offset in register A.
0447H - 0448H	Zero register B.
0449H	Add the offset in register pair BC to the starting address of the keyboard lookup table in register pair HL.
044AH	Load register A with the ASCII value at the location of the keyboard lookup table pointer in register pair HL.
044BH	Load register D with the ASCII value for the key pressed in register A.
044CH - 044EH	Load register pair BC with the delay count.
044FH - 0451H	Go delay.
0452H	Load register A with the ASCII value for the key pressed in register D.
0453H - 0454H	Check to see if the BREAK key was pressed.
0455H	Return if the BREAK key wasn't pressed.
0456H	Go if the BREAK key was pressed.
0457H	Return.
0458H - 058CH	**VIDEO DRIVER**
0458H - 045AH	Load register L with the LSB of the current cursor position at the location of the video device control block pointer in register pair IX plus three.
045BH - 045DH	Load register H with the MSB of the current cursor position at the location of the video device control block pointer in register pair IX plus four.
045EH - 045FH	Jump if get last character.
0460H - 0462H	Load register A with the cursor on/off flag at the

		location of the video device control block pointer in register pair IX plus five.
0463H		Check to see if the cursor is on or off.
0464H -	0465H	Jump if the cursor is off.
0466H		Display the character in register A at the location of the current cursor position in register pair HL.
0467H		Load register A with the character to be displayed in register C.
0468H -	0469H	Check to see if the character to be displayed in register A is a control code.
046AH -	046CH	Jump if the character to be displayed in register A is a control code.
046DH -	046EH	Check to see if the character to be displayed in register A is a graphic character or space compression code.
046FH -	0470H	Jump if the character to be displayed in register A is a graphic character or space compression code.
0471H -	0472H	Check to see if the character to be displayed in register A is an alphabetic character.
0473H -	0474H	Jump if the character to be displayed in register A is a nonalphabetic character.
0475H -	0476H	Adjust the alphabetic character in register A so that it will be in the range of from 0 to 25.
0477H -	0478H	Check to see if the character to be displayed in register A is a lower case character.
0479H -	047AH	Jump if the character to be displayed in register A isn't lower case.
047BH -	047CH	Convert the lower case character in register A to upper case.
047DH -	047FH	Go display the character in register A.
0480H		Load register A with the MSB of the current cursor position in register H.
0481H -	0482H	Mask the value in register A so that it will be in the range of video memory.
0483H -	0484H	Make sure that the value in register A will point to video memory.
0485H		Load register H with the updated value in register A.

0486H	Load register D with the value at the location of the current cursor position in register pair HL.
0487H - 0489H	Load register A with the cursor on/off flag at the location of the video device control block pointer in register pair IX plus five.
048AH	Check to see if the cursor is on or off.
048BH - 048CH	Jump if the cursor is off.
048DH - 048FH	Save the character being displayed in register D as the cursor on/off flag at the location of the video device control block pointer in register pair IX plus five.
0490H - 0491H	Display the cursor character at the current location of the cursor in register pair HL.
0492H - 0494H	Save the LSB of the current cursor position in register L at the location of the video device control block in register pair IX plus three.
0495H - 0497H	Save the MSB of the current cursor position in register H at the location of the video device control block in register pair IX plus four.
0498H	Load register A with the character that was displayed in register C.
0499H	Return.
049AH - 049CH	Load register A with the cursor on/off flag.
049DH	Check to see if the cursor is on or off.
049EH	Return if the cursor is on.
049FH	Load register A with the character at the location of the current cursor position in register pair HL.
04A0H	Return.
04A1H	Load register A with the LSB of the current position in register L.
04A2H - 04A3H	Mask the value in register A so that it will point to the beginning of the line.
04A4H	Load register L with the updated value in register A.
04A5H	Return.
04A6H - 04A7H	Check to see if the character to be displayed in register A is a space compression character.

04A8H - 04A9H	Jump if the character to be displayed in register A isn't a space compression character.
04AAH - 04ABH	Adjust the value in register A so that it will hold the number of spaces to be displayed.
04ACH - 04ADH	Jump if there aren't any spaces to be displayed.
04AEH	Load register B with the number of spaces to be displayed in register A.
04AFH - 04B0H	Load register A with a space character.
04B1H - 04B3H	Go display the character in register A.
04B4H - 04B5H	Loop till all of the spaces have been displayed.
04B6H - 04B7H	Jump.
04B8H	Load register A with the character being displayed at the location of the current cursor position in register pair HL.
04B9H - 04BBH	Save the value in register A as the cursor on/off flag at the location of the video device control block pointer in register pair IX plus five.
04BCH	Return.
04BDH	Zero register A.
04BEH - 04BFH	Jump.
04C0H - 04C2H	Load register pair HL with the starting address of video memory.
04C3H - 04C5H	Load register A with the 32/64 character per line flag.
04C6H - 04C7H	Adjust the value in register A for 64 characters per line.
04C8H - 04CAH	Save the value in register A as the 32/64 character per line flag.
04CBH - 04CCH	Send the value in register A out the cassette port.
04CDH	Return.
04CEH	Decrement the current cursor position in register pair HL.
04CFH - 04D1H	Load register A with the 32/64 character per line flag.
04D2H - 04D3H	Check to see if it's 32 or 64 characters per line.

04D4H - 04D5H	Jump if it's 64 characters per line.
04D6H	Decrement the current cursor position in register pair HL.
04D7H - 04D8H	Display a space character at the location of current cursor position in register pair HL.
04D9H	Return.
04DAH - 04DCH	Load register A with the 32/64 character per line flag.
04DDH - 04DEH	Check to see if it's 32 or 64 characters per line.
04DFH - 04E1H	Go if it's 32 characters per line.
04E2H	Load register A with the LSB of the current cursor position in register L.
04E3H - 04E4H	Mask the value in register A.
04E5H	Decrement the current cursor position in register pair HL.
04E6H	Return if still on the same line.
04E7H - 04E9H	Load register pair DE with the length of a line on the video display.
04EAH	Add the length of a line on the video display in register pair DE to the current cursor position in register pair HL.
04EBH	Return.
04ECH	Bump the current cursor position in register pair HL.
04EDH	Load register A with the LSB of the current cursor position in register L.
04EEH - 4EFH	Check to see if the cursor is still on the same line.
04F0H	Return if the cursor is still on the same line.
04F1H - 04F3H	Load register pair DE with a negative length of a line on the video display.
04F4H	Add the negative length of a line on the video display in register pair DE to the current cursor position in register pair HL.
04F5H	Return.

04F6H - 04F8H	Load register A with the 32/64 character per line flag.
04F9H - 04FAH	Adjust the value in register A for 32 characters per line.
04FBH - 04FDH	Save the value in register A as the 32/64 character per line flag.
04FEH - 04FFH	Send the value in register A out the cassette port.
0500H	Bump the current cursor position in register pair HL.
0501H	Load register A with the LSB of the current cursor position in register L.
0502H - 0503H	Make the value in register A an even value.
0504H	Load register L with the updated value in register A.
0505H	Return.
0506H - 0508H	Load register pair DE with the return address.
0509H	Save the return address in register pair DE on the stack.
050AH - 050BH	Check to see if the character in register A is a backspace and erase character.
050CH - 050DH	Jump if the character to be displayed in register A is a backspace cursor and erase character.
050EH - 050FH	Check to see if the character in register A is less than a line feed character.
0510H	Return if the character to be displayed in register A is less than a line feed character.
0511H - 0512H	Check to see if the character to be displayed in register A is less than or equal to a turn on the cursor character.
0513H - 0514H	Jump if the character to be displayed in register A is less than a turn on the cursor character so that a carriage return will be displayed.
0515H - 0516H	Jump if the character to be displayed in register A is a turn on the cursor character.
0517H - 0518H	Check to see if the character in register A is a turn off the cursor character.
0519H - 051AH	Jump if the character to be displayed in register A is a turn off the cursor character.

051BH	- 051CH	Check to see if the character to be displayed in register A is a turn on the 32 character per line mode character.
051DH	- 051EH	Jump if the character to be displayed in register A is a turn on the 32 character per line mode character.
051FH	- 0520H	Check to see if the character to be displayed in register A is a left arrow character.
0521H	- 0522H	Jump if the character to be displayed in register A is a left arrow character.
0523H	- 0524H	Check to see if the character to be displayed in register A is a right arrow character.
0525H	- 0526H	Jump if the character to be displayed in register A is a right arrow character.
0527H	- 0528H	Check to see if the character to be displayed in register A is a down arrow character.
0529H	- 052AH	Jump if the character to be displayed in register A is a down arrow character.
052BH	- 052CH	Check to see if the character to be displayed in register A is an up arrow character.
052DH	- 052EH	Jump if the character to be displayed in register A is an up arrow character.
052FH	- 0530H	Check to see if the character to be displayed in register A is a home the cursor character.
0531H	- 0532H	Jump if the character to be displayed in register A is a home the cursor character.
0533H	- 0534H	Check to see if the character to be displayed in register A is a backspace to the beginning of the line character.
0535H	- 0537H	Jump if the character to be displayed in register A is a backspace to the beginning of the line character.
0538H	- 0539H	Check to see if the character to be displayed in register A is an erase to the end of the line character.
053AH	- 053BH	Jump if the character to be displayed in register A is an erase to the end of the line character.
053CH	- 053DH	Check to see if the character to be displayed in register A is an erase to the end of the screen character.
053EH	- 053FH	Jump if the character to be displayed in register A is an erase to the end of the screen character.

0540H	Return if the character to be displayed isn't any of the above control codes.
0541H	Display the character in register A at the location of the current cursor position in register pair HL.
0542H	Bump the current cursor position in register pair HL.
0543H - 0545H	Load register A with the 32/64 character per line flag.
0546H - 0547H	Check to see if it's 32 or 64 characters per line.
0548H - 0549H	Jump if it's 64 characters per line.
054AH	Bump the current cursor position in register pair HL.
054BH	Load register A with the MSB of the current cursor position in register H.
054CH - 054DH	Check to see if the end of video memory plus one has been reached.
054EH	Return if the end of video memory plus one hasn't been reached.
054FH - 0551H	Load register pair DE with a negative length of a line on the video display.
0552H	Add the negative length of a line on the video display in register pair DE to the current cursor position in register pair HL.
0553H	Save the current cursor position in register pair HL on the stack.
0554H - 0556H	Load register pair DE with the starting address of video memory.
0557H - 0559H	Load register pair HL with the starting address of the second line of the video memory.
055AH	Save the value in register pair BC on the stack.
055BH - 055DH	Load register pair BC with the length of video memory to be moved.
0553H - 055FH	Move the last fifteen lines of video memory to the first fifteen lines of video memory.
0560H	Get the value from the stack and put it in register pair BC.
0561H	Load register pair HL with the starting address of the sixteenth line of video memory.

0562H - 0563H	Jump.
0564H	Load register A with the LSB of the current cursor position in register L.
0565H - 0566H	Adjust the value in register A so that it's the start of the current line.
0567H	Load register L with the updated value in register A.
0568H	Save the current cursor position in register pair HL on the stack.
0569H - 056BH	Load register pair DE with the length of a line on the video display.
056CH	Add the length of a line on the video display in register pair DE to the current cursor position in register pair HL.
056DH	Load register A with the MSB of the current cursor position in register H.
056EH - 056FH	Check to see if the end of video memory plus one has been reached.
0570H - 0571H	Jump if the end of video memory plus one has been reached.
0572H	Get the cursor position from the stack and put it in register pair DE.
0573H	Save the new cursor position in register pair HL on the stack.
0574H	Load register D with the MSB of the current cursor position in register H.
0575H	Load register A with the LSB of the current cursor position in register L.
0576H - 0577H	Mask the value in register A.
0578H	Save the updated value in register A in register E.
0579H	Bump the adjusted cursor position in register pair DE.
057AH - 057BH	Jump.
057CH	Save the cursor position in register pair HL on the stack.
057DH - 057FH	Load register pair DE with the end of video memory plus one.

0580H - 0581H	Display a space at the video memory pointer in register pair HL.
0582H	Bump the video memory pointer in register pair HL.
0583H	Load register A with the MSB of the video memory pointer in register H.
0584H	Check to see if the MSB of the video memory pointer in register A is the same as the MSB of the ending memory pointer in register D.
0585H - 0586H	Jump if the MSB of the video memory pointer in register A isn't the same as the MSB of the ending memory pointer in register D.
0587H	Load register A with the LSB of the video memory pointer in register L.
0588H	Check to see if the LSB of the video memory pointer in register A is the same as the LSB of the ending memory pointer in register E.
0589H - 058AH	Jump if the LSB of the video memory pointer in register A isn't the same as the LSB of the ending memory pointer in register E.
058BH	Get the current cursor position from the stack and put it in register pair HL.
058CH	Return.
058DH - 05D8H	**PRINTER DRIVER**
058DH	Load register A with the character to be sent to the printer in register C.
058EH	Check to see if the character to be sent to the printer is equal to zero.
058FH - 0590H	Jump if the character to be sent to the printer in register A is equal to zero.
0591H - 0592H	Check to see if the character to be sent to the printer in register A is a skip to the top of the form character.
0593H - 0594H	Jump if the character to be sent to the printer in register A is a skip to the top of the form character.
0595H - 0596H	Check to see if the character to be sent to the printer in register A is a conditional skip to the top of the form character.
0597H - 0598H	Jump if the character to be sent to the printer in

register A isn't a conditional skip to the top of the form character.

0599H — Zero register A.

059AH - 059CH — Check to see if the number of lines per page at the location of the printer device control block pointer in register pair IX plus three is the same as the value in register A.

059DH - 059EH — Jump if the number of lines per page at the location of the printer device control block pointer in register pair IX plus three is the same as the value in register A.

059FH - 05A1H — Load register A with the number of lines per page at the location of the printer device control block in register pair IX plus three.

05A2H - 05A4H — Subtract the lines printed so far at the location of the printer device control block pointer in register pair IX plus four from the number of lines per page in register A.

05A5H — Load register B with the number of lines to skip.

05A6H - 05A8H — Go check the printer status.

05A9H - 05AAH — Loop until the printer is ready.

05ABH - 05ACH — Load register A with a line feed character.

05ADH - 05AFH — Send the value in register A to the printer.

05B0H - 05B1H — Loop until all of the lines have been skipped.

05B2H - 05B3H — Jump.

05B4H — Save the character to be sent to the printer in register A on the stack.

05B5H - 05B7H — Go check the printer status.

05B8H - 05B9H — Loop until the printer is ready.

05BAH — Get the character to be sent to the printer from the stack and put it in register A.

05BBH - 05BDH — Send the character in register A to the printer.

05BEH - 05BFH — Check to see if the character sent to the printer was a carriage return.

0500H — Return if the character sent to the printer in register A isn't a carriage return.

05C1H - 05C3H	Bump the number of lines printed so far at the location of the printer device control block pointer in register pair IX plus four.
05C4H - 05C6H	Load register A with the number of lines printed so far at the location of the printer device control block pointer in register pair IX plus four.
05C7H - 05C9H	Check to see if the number of lines printed so far in register A is the same as the number of lines per page at the location of the printer device control block pointer in register pair IX plus three.
05CAH	Load register A with the character sent to the printer in register C.
05CBH	Return if the number of lines printed so far isn't the same as the number of lines per page.
05CCH - 05CFH	Zero the number of lines printed so far at the location of the printer device control block pointer in register pair IX plus four.
05D0H	Return.
05D1H - 05D3H	Get the value from the printer and put it in register A.
05D4H - 05D5H	Mask the value in register A.
05D6H - 05D7H	Check to see if the printer is ready.
05D8H	Return.
05D9H - 0673H	**INPUT ROUTINE**
05D9H	Save the start of the input buffer area pointer in register pair HL on the stack.
05DAH - 05DBH	Load register A with a turn on the cursor character.
05DCH - 05DEH	Go turn on the cursor.
05DFH	Load register C with the size of the input buffer in register B.
05E0H - 05E2H	Go wait till a key is pressed.
05E3H - 05E4H	Check to see if the key that was pressed in register A is greater than a space.
05E5H - 05E6H	Jump if the key that was pressed in register A is greater than or equal to a space.
05E7H - 05E8H	Check to see if the key that was pressed in register A is a carriage return.

05E9H - 05EBH	Jump if the key that was pressed in register A is a carriage return.
05ECH - 05EDH	Check to see if the key that was pressed in register A is the CLEAR key.
05EEH - 05EFH	Jump if the key that was pressed in register A is the CLEAR key.
05F0H - 05F1H	Check to see if the key that was pressed in register A is the BREAK key.
05F2H - 05F3H	Jump if the key that was pressed in register A is the BREAK key.
05F4H - 05F6H	Load register pair DE with the return address.
05F7H	Save the return address in register pair DE on the stack.
05F8H - 05F9H	Check to see if the key that was pressed in register A is a backspace the cursor and erase character.
05FAH - 05FBH	Jump if the key was pressed in register A is a backspace the cursor and erase character.
05FCH - 05FDH	Check to see if the key that was pressed in register A is a backspace character.
05FEH - 05FFH	Jump if the key that was pressed in register A is a backspace character.
0600H - 0601H	Check to see if the key that was pressed in register A is a tab character.
0602H - 0603H	Jump if the key that was pressed in register A is a tab character.
0604H - 0605H	Check to see if the key that was pressed in register A is a turn on the 32 character per line mode character.
0606H - 0607H	Jump if the key that was pressed in register A is a turn on the 32 character per line mode character.
0608H - 0609H	Check to see if the key that was pressed in register A is a line feed character.
060AH	Return if the key that was pressed in register A isn't a line feed character.
060BH	Get the return address from the stack and put it in register pair DE.
060CH	Save the key that was pressed in register A at the location of the input buffer pointer in register pair HL.

060DH	Load register A with the length of the buffer remaining in register B.
060EH	Check to see if there is anymore of the input buffer remaining.
060FH - 0610H	Jump if the end of the input buffer has been reached.
0611H	Load register A with the value at the location of the input buffer pointer in register pair HL.
0612H	Bump the input buffer pointer in register pair HL.
0613H - 0615H	Go display the character in register A.
0616H	Decrement the number of bytes remaining in the input buffer area in register B.
0617H - 0618H	Jump.
0619H - 061BH	Go clear the screen.
061CH	Load register B with the length of the input buffer in register C.
061DH	Get the starting address for the input buffer area from the stack and put it in register pair HL.
061EH	Save the starting address for the input buffer area in register pair HL on the stack.
061FH - 0621H	Jump.
0622H - 0624H	Go back up the input buffer pointer in register pair HL if necessary.
0625H	Decrement the input buffer pointer in register pair HL.
0626H	Load register A with the character at the location of the input buffer pointer in register pair HL.
0627H	Bump the input buffer pointer in register pair HL.
0628H - 0629H	Check to see if the character in register A is a line feed character.
062AH	Return if the character in register A is a line feed character.
062BH	Load register A with the number of bytes remaining in the input buffer area in register B.
062CH	Check to see if the number of characters remaining in the input buffer area in register A is the same as the length of the input buffer area in register C.

062DH - 062EH	Jump if there are characters in the buffer.
062FH	Return if the buffer is empty.
0630H	Load register A with the number of bytes remaining in the input buffer area in register B.
0631H	Check to see if the number of characters remaining in the input buffer area in register A is the same as the length of the input buffer area in register C.
0632H	Return if the input buffer area is empty.
0633H	Decrement the input buffer area pointer in register pair HL.
0634H	Load register A with the character at the location of the input buffer area pointer in register pair HL.
0635H - 0636H	Check to see if the character in register A is a line feed character.
0637H	Bump the input buffer area pointer in register pair HL.
0638H	Return if the character in register A is a line feed character.
0639H	Decrement the input buffer area pointer in register pair HL.
063AH - 063BH	Load register A with a backspace the cursor and erase character.
063CH - 063EH	Go backspace the cursor.
063FH	Bump the number of characters remaining in the input buffer area in register B.
0640	Return.
0641H - 0642H	Load register A with the turn on the 32 character per line mode character.
0643H - 0645H	Go turn on the 32 character per line mode.
0646H - 0648H	Go get the cursor line position and return with it in register A.
0649H - 064AH	Mask the cursor line position in register A.
064BH	Complement the value in register A.
064CH	Bump the value in register A.
064DH - 064EH	Adjust the value in register A so that it will hold the number of spaces to add.

064FH	Load register E with the number of spaces to be added in register A.
0650H	Load register A with the number of bytes remaining in the input buffer area in register B.
0651H	Check to see if there are no more bytes to be added.
0652H	Return if the input buffer is full.
0653H - 0654H	Load register A with a space character.
0655H	Save the space character in register A at the location of the input buffer area pointer in register pair HL.
0656H	Bump the input buffer area pointer in register pair HL.
0657H	Save the value in register pair DE on the stack.
0658H - 065AH	Go display the space character in register A.
065BH	Get the value from the stack and put it in register pair DE.
065CH	Decrement the number of bytes remaining in the input buffer area in register B.
065DH	Decrement the number of spaces to be added in register E.
065EH	Return if the number of spaces has been added to the input buffer.
065FH - 0660H	Loop till all the spaces have been added to the input buffer.
0661H	Set the Carry flag.
0662H	Save the value in register pair AF on the stack.
0663H - 0664H	Load register A with a carriage return character.
0665H	Save the carriage return character in register A at the location of the input buffer area pointer in register pair HL.
0666H - 0668H	Go display the carriage return character in register A.
0669H - 066AH	Load register A with a turn off the cursor character.
066BH - 066DH	Go display the turn off the cursor character in register A.
066EH	Load register A with the length of the input in register C.

066FH	Subtract the number of bytes remaining in the input buffer area in register B from the length of the input buffer area in register A.
0670H	Load register B with the number of characters in the input buffer area in register A.
0671H	Get the value from the stack and put it in register pair AF.
0672H	Get the starting address of the input buffer area from the stack and put it in register pair HL.
0673H	Return.
0674H - 06D1H	**INITIALIZATION ROUTINE**
0674H - 0675H	Send the zero in register A out the cassette port.
0676H - 0678H	Load register pair HL with the starting address of the ROM area to be moved to RAM.
0679H - 067BH	Load register pair DE with the starting address to move the data to in RAM.
067CH - 067EH	Load register pair BC with the length of the ROM area to be moved.
067FH - 0680H	Move the ROM area to RAM.
0681H	Decrement the value in register A.
0682H	Decrement the value in register A.
0683H - 0684H	Loop till register A is zero.
0685H - 0686H	Load register B with the number of bytes of memory to be zeroed.
0687H	Save the zero in register A at the location of the memory pointer in register pair DE.
0688H	Bump the memory pointer in register pair DE.
0689H - 068AH	Loop till all of the memory locations have been zeroed.
068BH - 068DH	Load register A with the location of keyboard memory for the BREAK key.
068EH - 068FH	Check to see if the BREAK key is being pressed.
0690H - 0692H	Jump if the BREAK key is not being pressed.
0693H - 0695H	Set the stack pointer to 407DH.

0696H	- 0698H	Load register A with the status of the disk controller.
0699H		Bump the value in register A.
069AH	- 069BH	Check to see if the disk controller is present.
069CH	- 069EH	Jump if the disk controller isn't present.
069FH	- 06A0H	Load register A with the drive to turn on.
06A1H	- 06A3H	Turn on drive 0.
06A4H	- 06A6H	Load register pair HL with the address of the disk command/status register.
06A7H	- 06A9H	Load register pair DE with the address of the disk data register.
06AAH	- 06ABH	Load the disk command register with the command to position the head to track 0.
06ACH	- 06AEH	Load register pair BC with the delay count.
06AFH	- 06B1H	Go delay.
06B2H	- 06B3H	Check to see if the disk is busy.
06B4H	- 06B5H	Loop till the disk is no longer busy.
06B6H		Zero register A.
06B7H	- 06B9H	Save the value in register A in the disk sector register.
06BAH	- 06BCH	Load register pair BC with the address in memory to place the sector read.
06BDH	- 06BEH	Load register A with the command to read the sector.
06BFH		Put the command in register A in the disk command register.
06C0H	- 06C1H	Check to see if there is data available.
06C2H	- 06C3H	Loop till there is data available.
06C4H		Load register A with the byte read from the disk.
06C5H		Save the value in register A at the location of the memory pointer in register pair BC.
06C6H		Bump the LSB of the memory pointer in register C.
06C7H	- 06C8H	Loop till the whole of the sector has been read.

06C9H - 06CBH Jump to the TRSDOS loader routine just loaded.

06CCH - 06CEH Load register pair BC with the starting address of the Level II BASIC READY routine.

06CFH - 06D1H Jump.

06D2H - 0707H **ROM STORAGE LOCATION FOR DATA TO BE MOVED TO RAM BY THE INITIALIZATION PROCESS.**

0708H - 070AH Load register pair HL with the starting address of a single precision value stored in ROM.

070BH - 070FH **SINGLE PRECISION ADDITION, REG1 = (HL) + REG1**

070BH - 070DH Load the single precision value pointed to by register pair HL into register pairs BC and DE.

070EH - 070FH Jump.

0710H - 0712H **SINGLE PRECISION SUBTRACTION, REG1 = (HL) – REG1**

0710H - 0712H Go load the single precision value pointed to by register pair HL into register pairs BC and DE.

0713H - 0715H **SINGLE PRECISION SUBTRACTION, REG1 = BCDE – REG1**

0713H - 0715H Go reverse the sign of the single precision value in register pairs BC and DE.

0716H - 0752H **SINGLE PRECISION ADDITION, REG1 = BCDE + REG1**

0716H Load register A with the exponent of the single precision value in register B.

0717H Check to see if the single precision value in register pairs BC and DE is equal to zero.

0718H Return if the single precision value in register pairs BC and DE is equal to zero.

0719H - 071BH Load register A with the exponent of the single precision value in REG1.

071CH Check to see if the single precision value in REG1 is equal to zero.

071DH - 071FH Jump if the single precision value in REG1 is equal to zero.

0720H Subtract the value of the exponent for the single

precision value in register B from the value of the exponent for the single precision value in REG1 in register A.

0721H - 0722H Jump if the single precision value in register pairs BC and DE is smaller than the single precision value in REG1.

0723H Adjust the difference in the exponents in register A so that it is positive.

0724H Bump the difference in the exponents in register A so that it will be the correct positive number.

0725H Load register pair HL with the 16-bit value in register pair DE.

0726H - 0728H Go put the single precision value in REG1 on the stack.

0729H Load register pair DE with the 16-bit value in register pair HL.

072AH - 072CH Go put the single precision value in register pairs BC and DE into REG1.

072DH Get the 16-bit value from the stack and put it in register pair BC.

072EH Get the 16-bit value from the stack and put it in register pair DE.

072FH - 0730H Check to see if the difference in the exponents in register A is greater than 18H.

0731H Return if the difference in the exponents is too great.

0732H Save the difference in the exponents in register A on the stack.

0733H - 0735H Set the sign bits for the single precision values and return with the equality of the sign bits in register A.

0736H Load register H with the equality of the sign bits in register A.

0737H Get the difference of the exponents from the stack and put it in register A.

0738H - 073AH Go shift the single precision value in register pairs BC and DE till it lines up with the single precision value in REG1.

073BH Check to see if the sign bits are equal.

073CH - 073EH Load register pair HL with the starting address of REG1.

073FH - 0741H	Jump if the signs aren't equal.
0742H - 0744H	Go add the single precision value in BCDE to the single precision value in REG1.
0745H - 0747H	Jump if the exponent remains unchanged.
0748H	Bump the memory pointer in register pair HL, so that it points to the exponent in REG1.
0749H	Bump the exponent in REG1 at the location of the memory pointer in register pair HL.
074AH - 074CH	Go to the Level II BASIC error routine and output an OV ERROR message if the exponent in REG1 is too large.
074DH - 074EH	Load register L with the number of bits to shift the single precision result in register pairs BC and DE.
074FH - 0751H	Go shift the single precision result in register pairs BC and DE.
0752 - 0753H	Jump.
0754H - 077CH	**SINGLE PRECISION MATH ROUTINE**
0754H	Zero register A.
0755H	Subtract the 8-bit value in register B from the value in register A.
0756H	Load register B with the result in register A.
0757H	Load register A with the value at the memory pointer in register pair HL.
0758H	Subtract the value in register E from the value in register A.
0759H	Load register E with the result in register A.
075AH	Bump the memory pointer in register pair HL.
075BH	Load register A with the value at the location of the memory pointer in register pair HL.
075CH	Subtract the value in register D from the value in register A.
075DH	Load register D with the result in register A.
075EH	Bump the memory pointer in register pair HL.
075FH	Load register A with the value at the location of the memory pointer in register pair HL.

0760H	Subtract the value in register C from the value in register A.
0761H	Load register C with the result in register A.
0762H - 0764H	If the Carry flag is set, go convert the single precision value to a positive number.
0765H	Load register L with the exponent of the original value in register pairs BC and DE.
0766H	Load register H with the LSB of the single precision value in register E.
0767H	Zero register A.
0768H	Load register B with the new exponent in register A.
0769H	Load register A with the MSB of the single precision value in register C.
076AH	Check to see if the value in register A is equal to zero.
076BH - 076CH	Jump if the MSB of the single precision value is nonzero.
076DH	Shift the NMSB into the MSB by loading register C with the value in register D.
076EH	Shift the LSB into the NMSB by loading register D with the value in register H.
076FH	Load register H with the value in register L.
0770H	Load register L with the value in register A.
0771H	Load register A with the new exponent counter in register B.
0772H - 0773H	Subtract the number of bits just shifted from the new exponent counter in register A.
0774H - 0775H	Check to see if three bytes have been shifted.
0776H - 0777H	Loop till shift is completed.
0778H	Zero register A.
0779H - 077BH	Save the value in register A as the exponent of the single precision result in REG1.
077CH	Return with a single precision value of zero in REG1.
077DH - 07A7H	**SINGLE PRECISION MATH ROUTINE**

077DH	Decrement the new exponent counter in register B.
077EH	Shift the 16-bit value in register pair HL left one bit.
077FH	Load register A with the NMSB in register D.
0780H	Shift the NMSB in register A left one bit and shift a bit from register pair HL if necessary.
0781H	Save the adjusted NMSB in register A into register D.
0782H	Load register A with the MSB in register C.
0783H	Shift the MSB in register A left one bit and shift a bit from register D if necessary.
0784H	Load register C with the adjusted value in register A.
0785H - 0787H	Loop till the most significant bit of the single precision value is equal to one.
0788H	Load register A with the new exponent counter in register B.
0789H	Load register E with the LSB of the single precision value in register H.
078AH	Load register B with the value in register L.
078BH	Check to see if there were any bits shifted.
078CH - 078DH	Jump if there weren't any bits shifted.
078EH - 0790H	Load register pair HL with the address of the exponent in REG1.
0791H	Add the value of the original exponent at the location of the memory pointer in register pair HL to the number of bits shifted in register A.
0792H	Save the new exponent in register A at the location of the memory pointer in register pair HL.
0793H - 0794H	Jump if exponent is too small.
0795H	Return if exponent is equal to zero.
0796H	Load register A with the LSB of the single precision value in register B.
0797H - 0799H	Load register pair HL with the address of the exponent in REG1.
079AH	Check to see if the most significant bit of the value in register A is set.

079BH - 079DH	Go check for overflow if the most significant bit in the value in register A is set.
079EH	Load register B with the exponent at the location of the memory pointer in register pair HL.
079FH	Bump the memory pointer in register pair HL.
07A0H	Load register A with the value of the sign at the location of the memory pointer in register pair HL.
07A1H - 07A2H	Mask the sign bit in register A.
07A3H	Set the sign bit in register A.
07A4H	Load register C with the adjusted MSB of the single precision value in register A.
07A5H - 07A7H	Jump.
07A8H - 07B6H	**SINGLE PRECISION MATH ROUTINE**
07A8H	Bump the LSB of the single precision value in register E.
07A9H	Return if the adjusted LSB of the single precision value in register E is nonzero.
07AAH	Bump the NMSB of the single precision value in register D.
07ABH	Return if the adjusted NMSB of the single precision value in register D is nonzero.
07ACH	Bump the MSB of the single precision value in register C.
07ADH	Return if the adjusted MSB of the single precision value in register C is nonzero.
07AEH - 07AFH	Adjust the MSB of the single precision value in register C.
07B0H	Bump the exponent of the single precision value at the location of the memory pointer in register pair HL.
07B1H	Return if the adjusted exponent of the single precision value at the location of the memory pointer in register pair HL is nonzero.
07B2H - 07B3H	Load register E with an OV ERROR code.
07B4H - 07B6H	Go to the Level II BASIC error routine and display an OV ERROR message if the value has overflowed.

07B7H - 07C2H SINGLE PRECISION MATH ROUTINE

07B7H	Load register A with the LSB of the single precision value at the location of the memory pointer in register pair HL.
07B8H	Add the LSB of the single precision value in register E to the LSB of the single precision value in register A.
07B9H	Load register E with the result in register A.
07BAH	Bump the memory pointer in register pair HL.
07BBH	Load register A with the NMSB of the single precision value in REG1 at the location of the memory pointer in register pair HL.
07BCH	Add the NMSB of the single precision value in register D to the NMSB of the single precision value in register A.
07BDH	Load register D with the result in register A.
07BEH	Bump the memory pointer in register pair HL.
07BFH	Load register A with the MSB of the single precision value in REG1 at the location of the memory pointer in register pair HL.
07C0H	Add the MSB of the single precision value in register C to the MSB of the single precision value in register A.
07C1H	Load register C with the result in register A.
07C2H	Return.

07C3H - 07D6H SINGLE PRECISION MATH ROUTINE

07C3H - 07C5H	Load register pair HL with the address of the sign flag storage location.
07C6H	Load register A with the value of the sign flag at the location of the memory pointer in register pair HL.
07C7H	Complement the sign flag in register A.
07C8H	Save the adjusted sign flag in register A at the location of the memory pointer in register pair HL.
07C9H	Zero register A.
07CAH	Load register L with the value in register A.
07CBH	Figure the negative value for register B by sub-

	tracting the current value in register B from the value in register A.
07CCH	Save the adjusted value in register A in register B.
07CDH	Load register A with zero.
07CEH	Figure the negative LSB of the single precision value in register E by subtracting the current LSB of the single precision value in register E from the value in register A.
07CFH	Load register E with the adjusted LSB of the single precision value in register A.
07D0H	Load register A with zero.
07D1H	Figure the negative NMSB of the single precision value in register D by subtracting the current NMSB of the single precision value in register D from the value in register A.
07D2H	Load register D with the adjusted NMSB of the single precision value in register A.
07D3H	Load register A with zero.
07D4H	Figure the negative MSB of the single precision value in register C by subtracting the current MSB of the single precision value in register C from the value in register A.
07D5H	Load register C with the adjusted MSB of the single precision value in register A.
07D6H	Return.
07D7H - 07F7H	**SINGLE PRECISION MATH ROUTINE**
07D7H - 07D8H	Load register B with zero.
07D9H - 07DAH	Check to see if the shift counter in register A still indicates at least 8 bits have to be shifted.
07DBH - 07DCH	Jump if less than 8 bits are left to be shifted.
07DDH	Load register B with the LSB of the single precision value in register E.
07DEH	Load register E with the NMSB of the single precision value in register D.
07DFH	Load register D with the MSB of the single precision value in register C.
07E0H - 07E1H	Load register C with zero.

07E2H - 07E3H	Loop till there is less than 8 bits left to be shifted.
07E4H - 07E5H	Adjust the shift counter in register A to its correct value.
07E6H	Load register L with the shift counter in register A.
07E7H	Zero register A.
07E8H	Decrement the shift counter in register L.
07E9H	Return if there are no more bits to be shifted.
07EAH	Load register A with the MSB of the single precision value in register C.
07EBH	Shift the MSB of the single precision value in register A one place to the right.
07ECH	Load register C with the adjusted MSB of the single precision value in register A.
07EDH	Load register A with the NMSB of the single precision value in register D.
07EEH	Shift the NMSB of the single precision value in register A one place to the right and pick up the value of the Carry flag.
07EFH	Load register D with the adjusted NMSB of the single precision value in register A.
07F0H	Load register A with the LSB of the single precision value in register E.
07F1H	Shift the LSB of the single precision value in register A one place to the right and pick up the value of the Carry flag.
07F2H	Load register E with the adjusted LSB of the single precision value in register A.
07F3H	Load register A with the value in register B.
07F4H	Shift the value in register A one place to the right and pick up the value of the Carry flag.
07F5H	Load register B with the adjusted value in register A.
07F6H - 07F7H	Loop till all of the bits have been shifted.
07F8H - 07FBH	**SINGLE PRECISION CONSTANT STORAGE LOCATION**

07F8H - 07FBH A single precision constant equal to 1.0 is stored here.

07FCH - 0808H **SINGLE PRECISION CONSTANTS STORAGE LOCATION**

07FCH The number of single precision constants which follows is stored here.

07FDH - 0800H A single precision constant equal to .598978 is stored here.

0801H - 0804H A single precision constant equal to .961471 is stored here.

0805H - 0808H A single precision constant equal to 2.88539 is stored here.

0809H - 0846H **LEVEL II BASIC LOG ROUTINE**

0809H - 080BH Go check the sign of the single precision value in REG1.

080CH Check to see if the single precision value in REG1 is negative or positive.

080DH - 080FH Go the Level II BASIC error routine and display a FC ERROR message if the current single precision value in REG1 is negative.

0810H - 0812H Load register pair HL with the address of the exponent in REG1.

0813H Load register A with the exponent of the single precision value in REG1 at the location of the memory pointer in register pair HL.

0814H - 0816H Load register BC with the exponent and the MSB of a single precision constant.

0817H - 0819H Load register DE with the NMSB and the LSB of a single precision constant. Register pairs BC and DE are now equal to the single precision constant of .707107.

081AH Subtract the exponent in register B from the exponent of the x-value in register A.

081BH Save the difference between the two exponents in register A on the stack.

081CH Save the exponent in register B as the exponent of the x-value in REG1 at the location of the memory pointer in register pair HL.

081DH	Save the NMSB and the LSB of the single precision value in register pair DE on the stack.
081EH	Save the exponent and the MSB of the single value in register pair BC on the stack.
081FH - 0831H	Go add the x-value to the single precision constant in register pairs BC and DE and return with the result in REG1.
0822H	Get the exponent and the MSB of the single precision value from the stack and put it in register pair BC.
0823H	Get the NMSB and the LSB of the single precision value from the stack and put it in register pair DE.
0824H	Multiply the single precision value in register pairs BC and DE by two by bumping the exponent in register B.
0825H - 0827H	Go divide the single precision value in register pairs BC and DE by the x-value in REG1 and return with the result in REG1.
0828H - 082AH	Load register pair HL with the starting address of a single precision constant.
082BH - 082DH	Go subtract the x-value in REG1 from the single precision constant of 1.0 at the location of the memory pointer in register pair HL and return with the result in REG1.
082EH - 0830H	Load register pair HL with the starting address of a storage location for single precision constants to be used for a series of computations.
0831H - 0833H	Go do a series of computations and return with the result in REG1.
0834H - 0836H	Load register BC with the exponent and the MSB of a single precision constant.
0837H - 0839H	Load register pair DE with the NMSB and the LSB of a single precision. Register pairs BC and DE are now equal to a single precision of −0.5.
083AH - 083CH	Go add the x-value in REG1 to the single precision constant in register pairs BC and DE and return with the result in REG1.
083DH	Get the difference between the two original exponents from the stack and put it in register A.
083EH - 0840H	Go convert the value in register A to a single precision number and add it to the x-value in REG1.

Return with the result in REG1.

0841H	- 0843H	Load register pair BC with the exponent and the MSB of a single precision constant.
0844H	- 0846H	Load register pair DE with the NMSB and the LSB of a single precision constant. Register pairs BC and DE are now equal to a single precision value of 0.693147.
0847H	- 0891H	**SINGLE PRECISION MULTIPLICATION, REG1 = BCDE * REG1**
0847H	- 0849H	Go check to see if the single precision value in REG1 is equal to zero.
084AH		Return if the single precision value in REG1 is equal to zero.
084BH	- 084CH	Load register L with a bit mask.
084DH	- 084FH	Go check for possible overflow and the single precision value in register pairs BC and DE equal to zero.
0850H		Load register A with the single precision value's MSB in register C.
0851H	- 0853H	Save the MSB of the single precision value in register A at memory location 414FH.
0854H		Load register pair HL with the NMSB and the LSB of the single precision value in register pair DE.
0855H	- 0857H	Save the NMSB and the LSB of the single precision value in register pair HL at memory locations 4150H and 4151H.
0858H	- 085AH	Load register pair BC with zero.
085BH		Load register D with the value in register B.
085CH		Load register E with the value in register B.
085DH	- 085FH	Load register pair HL with the return address.
0860H		Save the return address in register pair HL on the stack.
0861H	- 0863H	Load register pair HL with the return address.
0864H		Save the return address in register pair HL on the stack.
0865H		Save the return address in register pair HL on the stack.

0866H - 0868H	Load register pair HL with the starting address of the single precision value in REG1.
0869H	Load register A with the LSB of the single precision value in REG1 at the location of the memory pointer in register pair HL.
085AH	Bump the memory pointer in register pair HL.
085BH	Check to see if the LSB of the single precision value in REG1 in register A is equal to zero.
086CH - 086DH	Jump if the LSB of the single precision value in REG1 is equal to zero.
086EH	Save the memory pointer in register HL on the stack.
086FH - 0870H	Load register L with the bit shift counter.
0871H	Shift the LSB of the single precision value in REG1 in register A one place to the right.
0872H	Load register H with the adjusted LSB in register A.
0873H	Load register A with the MSB of the single precision value in register C.
0874H - 0875H	Jump if bit 0 in the LSB just shifted wasn't set.
0876H	Save the value in register pair HL on the stack.
0877H - 0879H	Load register pair HL with the NMSB and the LSB of the original value in register pairs BC and DE.
087AH	Add the NMSB and the LSB of the total figured so far in register pair DE to the NMSB and the LSB of the original value in register pair HL.
087BH	Load register pair DE with the adjusted total in register pair HL.
087CH	Get the value from the stack and put it in register pair HL.
087DH - 087FH	Load register A with the MSB of the original value in register pairs BC and DE.
0880H	Add the MSB of the original value in register A to the MSB of the total figured so far in register C.
0881H	Shift the adjusted MSB of the total in register A one place to the right.
0882H	Load register C with the adjusted MSB of the total in register A.

0883H	Load register A with the NMSB of the total in register D.
0884H	Shift the NMSB of the total in register A one place to the right.
0885H	Load register D with the adjusted NMSB of the total in register A.
0886H	Load register A with the LSB of the total in register E.
0887H	Shift the LSB of the total in register A one place to the right.
0888H	Load register E with the adjusted LSB of the total in register A.
0889H	Load register A with the value in register B.
088AH	Shift the value in register A one place to the right.
088BH	Load register B with the adjusted value in register A.
088CH	Decrement the bit counter in register L.
088DH	Load register A with the LSB of the current value in register H.
088EH - 088FH	Loop till 8 bits have been shifted.
0890H	Get the memory pointer from the stack and put it in register pair HL.
0891H	Return.
0892H - 0896H	**SINGLE PRECISION MATH ROUTINE**
0892H	Load register B with the LSB of the single precision value in register E.
0893H	Load register E with the NMSB of the single precision value in register D.
0894H	Load register D with the MSB of the single precision value in register C.
0895H	Load register C with the value in register A.
0896H	Return.
0897H - 08A1H	**SINGLE PRECISION MATH ROUTINE**
0897H - 0899H	Go move the single precision value in REG1 onto the stack.

089AH - 089CH	Load register pair HL with the starting address of a single precision constant equal to 10.
089DH - 089FH	Go move the single precision value pointed to by register pair HL into REG1.
08A0H	Get the exponent and the MSB of the single precision value on the stack and put it in register pair BC.
08A1H	Get the NMSB and the LSB of the single precision value from the stack and put it in register pair DE.
08A2H - 0903H	**SINGLE PRECISION DIVISION,** REG1 = BCDE / REG1
08A2H - 08A4H	Go check to see if the single precision value in REG1 is equal to zero.
08A5H - 08A7H	Go to the Level II BASIC error routine and display an /0 ERROR message if the single precision value in REG1 is equal to zero.
08A8H - 08A9H	Load register L with a bit mask.
08AAH - 08ACH	Go adjust the exponent in REG1 for division.
08ADH	Bump the value of the exponent for the single precision value in REG1 at the location of the memory pointer in register pair HL.
08AEH	Bump the value of the exponent for the single precision value in REG1 at the location of the memory pointer in register pair HL.
08AFH	Decrement the value of the memory pointer in register pair HL.
08B0H	Load register A with the MSB of the single precision value in REG1 at the location of the memory pointer in register pair HL.
08B1H - 08B3H	Save the MSB of the single precision value in REG1 in register A at memory location 4089H.
08B4H	Decrement the value of the memory pointer in register pair HL.
08B5H	Load register A with the NMSB of the single precision value in REG1 at the location of the memory pointer in register pair HL.
08B6H - 08B8H	Save the NMSB of the single precision value in REG1 in register A at memory location 4085H.
08B9H	Decrement the value of the memory pointer in register pair HL.

08BAH	Load register A with the LSB of the single precision value in REG1 at the location of the memory pointer in register pair HL.
08BBH - 08BDH	Save the LSB of the single precision value in REG1 in register A at memory location 4081H.
08BEH	Load register B with the MSB of the single precision value in register C.
08BFH	Load register pair HL with the NMSB and the LSB of the single precision value in register pair DE.
08C0H	Zero register A.
08C1H	Zero the MSB of the total by loading register C with the value in register A.
08C2H	Zero the NMSB of the total by loading register D with the value in register A.
08C3H	Zero the LSB of the total by loading register E with the value in register A.
08C4H - 08C6H	Zero memory location 408CH.
08C7H	Save the NMSB and LSB of the dividend in register pair HL on the stack.
08C8H	Save the MSB of the dividend in register B on the stack.
08C9H	Load register A with the LSB of the dividend in register L.
08CAH - 08CCH	Go to the Level II BASIC division routine.
08CDH - 08CEH	Adjust the flags.
08CFH	Invert the Carry flag.
08D0H - 08D1H	Jump if not done.
08D2H - 08D4H	Save the value in register A at memory location 408CH.
08D5H	Get the value from the stack and put it in register pair AF.
08D6H	Get the value from the stack and put it in register pair AF.
08D7H	Set the Carry flag.
08D9H	Get the value from the stack and put it in register pair BC.

08DAH	Get the value from the stack and put it in register pair HL.
08DBH	Load register A with the MSB of the total in register C.
08DCH	Bump the MSB of the total in register A.
08DDH	Decrement the MSB of the total in register A.
08DEH	Shift the MSB of the total in register A one place to the right.
08DFH - 08E1H	Jump if done.
08E2H	Reset the Carry flag.
08E3H	Load register A with the LSB of the total in register E.
08E4H	Multiply the LSB of the total in register A by two.
08E5H	Load register E with the adjusted LSB of the total in register A.
08E6H	Load register A with the NMSB of the total in register D.
08E7H	Multiply the NMSB of the total in register A by two.
08E8H	Load register D with the adjusted NMSB of the total in register A.
08E9H	Load register A with the MSB of the total in register C.
08EAH	Multiply the MSB of the total in register A by two.
08EBH	Load register C with the adjusted MSB of the total in register A.
08ECH	Multiply the NMSB and the LSB of the dividend in register pair HL by two.
08EDH	Load register A with the MSB of the dividend in register B.
08EEH	Multiply the MSB of the dividend in register A by two.
08EFH	Load register B with the adjusted MSB of the dividend in register A.
08F0H - 08F2H	Load register A with the value at memory location 408CH.

08F3H	Multiply the value in register A by two.
08F4H - 08F6H	Save the adjusted value in register A at memory location 408CH.
08F7H	Load register A with the MSB of the total in register C.
08F8H	Combine the NMSB of the total in register D with the value in register A.
08F9H	Combine the LSB of the total in register E with the value in register A.
08FAH - 08FBH	Jump if the total isn't equal to zero.
08FCH	Save the NMSB and the LSB of the dividend in register pair HL on the stack.
08FDH - 08FFH	Load register pair HL with the address of the exponent in REG1.
0900H	Decrement the exponent in REG1 at the location of the memory pointer in register pair HL.
0901H	Get the NMSB and the LSB of the dividend from the stack and put it in register pair HL.
0902H - 0903H	Jump if the exponent in REG1 isn't equal to zero.
0904H - 0906H	**DISPLAY OV ERROR MESSAGE**
0904H - 0906H	Go to the Level II BASIC error routine and display an OV ERROR message.
0907H - 0913H	**DOUBLE PRECISION MATH ROUTINE**
0907H - 0908H	Load register A with a bit mask.
090AH	Load register A with a bit mask.
090BH - 090DH	Load register pair HL with the address of the MSB in REG2.
090EH	Load register C with the MSB of the value in REG2 at the location of the memory pointer in register pair HL.
090FH	Bump the value of the memory pointer in register pair HL.
0910H	Combine the value of the exponent in REG2 at the location of the memory pointer in register pair HL with the bit mask in register A.

0911H	Load register B with the adjusted exponent in register A.
0912H - 0913H	Load register L with a bit mask.
0914H - 0930H	**SINGLE PRECISION MATH ROUTINE**
0914H	Load register A with the exponent in register B.
0915H	Check to see if the exponent in register A is equal to zero.
0916H - 0917H	Jump if the exponent in register A is equal to zero.
0918H	Load register A with the bit mask in register L.
0919H - 091BH	Load register pair HL with the address of the exponent in REG1.
091CH	Combine the value of the exponent at the location of the memory pointer in register pair HL with the bit mask in register A.
091DH	Add the value of the exponent in register B to the value of the exponent in register A.
091EH	Load register B with the combined exponents in register A.
091FH	Shift the value of the combined exponents in register A one place to the right.
0920H	Check to see if the Carry flag was set by combining the two exponents.
0921H	Load register A with the combined exponents value in register B.
0922H - 0924H	Jump if overflow has occurred.
0925H - 0926H	Adjust the value of the combined exponents to it's proper value.
0927H	Save the value of the combined exponent in register A as the exponent in REG1 at the location of the memory pointer in register pair HL.
0928H - 092AH	Jump if the combined exponent in register A is equal to zero.
092BH - 092DH	Go turn on the sign bit of the MSB in REG1 and register B and save the sign bits.
092EH	Save the value in register A at the location of the memory pointer in register pair HL.

092FH	Decrement the memory pointer in register pair HL so that it points to the exponent in REG1.
0930H	Return.
0931H - 093DH	**SINGLE PRECISION MATH ROUTINE**
0931H - 0933H	Go check the value of the sign bit for the value in REG1.
0934H	Reverse the result of the sign bit test in register A.
0935H	Get the value from the stack and put it in register HL.
0936H	Set the flags according to the value of the sign bit test.
0937H	Get the value from the stack and put it in register pair HL.
0938H - 093AH	Jump if the value in REG1 is negative.
093BH - 093DH	Jump.
093EH - 0954H	**SINGLE PRECISION MATH ROUTINE**
093EH - 0940H	Go move the single precision value in REG1 to register pairs BC and DE.
0941H	Load register A with the value of the exponent in register B.
0942H	Check to see if the exponent in register A is equal to zero.
0943H	Return if the single precision value in register pairs BC and DE is equal to zero.
0944H - 0945H	Multiply the value of the exponent in register A by four.
0946H - 0948H	Go to the Level II BASIC error routine and display an OV ERROR routine if the adjusted exponent in register A is too large.
0949H	Load register B with the adjusted exponent in register A.
094AH - 094CH	Go add the original value in REG1 to the adjusted value in register pairs BC and DE and return with the original result in REG1. REG1 now holds the original value times five.
094DH - 094FH	Load register pair HL with the address of the exponent in REG1.

0950H	Bump the value of the exponent in REG1 at the location of the memory pointer in register pair HL. REG1 now holds the original value times ten.
0951H	Return if the new value in REG1 is in an acceptable range.
0952H - 0954H	Go to the Level II BASIC error routine and display an OV ERROR message if the value of the exponent at the location of the memory pointer in register pair HL is too large.
0955H - 0963H	**SINGLE PRECISION MATH ROUTINE**
0955H - 0957H	Load register A with the value of the exponent in REG1.
0958H	Check to see if the exponent in register A is equal to zero.
0959H	Return if the single precision value in REG1 is equal to zero.
095AH - 095CH	Load register A with the MSB of the single precision value in REG1.
095EH	Reverse the value in register A.
095FH	Put the value of the sign bit in register A into the Carry flag.
0960H	Make register A equal to −1 if the sign bit is negative or a value of 0 if the sign bit is positive.
0961H	Return if the single precision value in REG1 is negative.
0962H	Bump the value in register A so that register A will be equal to 1 if the single precision value in REG1 is positive.
0963H	Return.
0964H - 0976H	**SINGLE PRECISION MATH ROUTINE**
0964H - 0965H	Load register B with an exponent for an integer value.
0966H - 0968H	Load register pair DE with zero.
0969H - 096BH	Load register pair HL with the address of the exponent in REG1.
096CH	Load register C with the MSB of the integer value.
096DH	Save the exponent in register B in REG1 at the

	location of the memory pointer in register pair HL.
096EH - 096FH	Load register B with zero.
0970H	Bump the memory pointer in register pair HL.
0971H - 0972H	Save the sign value at the location of the memory pointer in register pair HL.
0973H	Shift the value of the sign bit for the MSB in register A into the Carry flag.
0974 - 0976H	Jump.
0977H - 0989H	**LEVEL II BASIC MATH ROUTINE**
0977H - 0979H	Go determine the sign of the current value in REG1.
097AH	Return if the value in REG1 is positive.
097BH	Go check the value of the current number type flag.
097CH - 097EH	Jump if the current value in REG1 is an integer.
097FH - 0981H	Go to the Level II BASIC error routine and display a TM ERROR message if the current value in REG1 is a string.
0982H - 0984H	Load register pair HL with the address of the MSB in REG1.
0985H	Load register A with the MSB in REG1 at the location of the memory pointer in register pair HL.
0986H - 0987H	Set the sign bit in the MSB in register A.
0988H	Save the adjusted MSB in register A in REG1 at the location of the memory pointer in register pair HL.
0989H	Return.
098AH - 0993H	**LEVEL II BASIC MATH ROUTINE**
098AH - 098CH	Go test the sign of the current value in REG1.
098DH	Load register L with the result of the sign test in register A.
098EH	Shift the sign bit in register A into the Carry flag.
098FH	Adjust the value in register A so that it will be equal to zero if the current value in REG1 is positive and equal to −1 if the current value in REG1 is negative.
0990H	Save the adjusted value in register A in register L.
0991H - 0993H	Jump.

0994H - 09A3H	**LEVEL II BASIC MATH ROUTINE**
0994H	Go check the current value of the number type flag.
0995H - 0997H	Go to the Level II BASIC error routine and display a TM ERROR message if the current value in REG1 is a string.
0998H - 099AH	Jump if the current value in REG1 is single precision or double precision.
099BH - 099DH	Load register pair HL with the integer value in REG1.
099EH	Load register A with the MSB of the integer value in register H.
099FH	Check to see if the integer value in REG1 is equal to zero.
09A0H	Return if the integer value in REG1 is equal to zero.
09A1H	Load register A with the MSB of the integer value in register H.
09A2H - 09A3H	Jump.
09A4H - 09B0H	**SINGLE PRECISION MATH ROUTINE**
09A4H	Load register pair DE with the value in register pair HL.
09A5H - 09A7H	Load register pair HL with the LSB and the NMSB of the single precision value in REG1.
09A8H	Exchange the return address on the stack with the NMSB and the LSB of the single precision value in register pair HL.
09A9H	Save the return address in register pair HL on the stack.
09AAH - 09ACH	Load register pair HL with the exponent and the MSB of the single precision value in REG1.
09ADH	Exchange the return address on the stack with the exponent and the MSB of the single precision value in register pair HL.
09AEH	Save the return address in register pair HL on the stack.
09AFH	Load register pair HL with the value in register pair DE.
09B0H	Return.

09B1H - 09BEH	**SINGLE PRECISION MATH ROUTINE**
09B1H - 09B3H	Go put the single precision value pointed to by register pair HL into register pairs BC and DE.
09B4H	Load register pair HL with the NMSB and the LSB of the single precision value in register pair DE.
09B5H - 09B7H	Save the NMSB and the LSB of the single precision value in register pair HL in REG1.
09B8H	Load register H with the exponent of the single precision value in register B.
09B9H	Load register L with the MSB of the single precision value in register C.
09BAH - 09BCH	Save the exponent and the MSB of the single precision value in register pair HL in REG1.
09BDH	Load register pair HL with the value in register pair DE.
09BEH	Return.
09BFH - 09CAH	**SINGLE PRECISION MATH ROUTINE**
09BFH - 09C1H	Load register pair HL with the starting address for a single precision value in REG1.
09C2H	Load register E with the LSB of the single precision value in REG1 at the location of the memory pointer in register pair HL.
09C3H	Bump the value of the memory pointer in register pair HL.
09C4H	Load register D with the NMSB of the single precision value in REG1 at the location of the memory pointer in register pair HL.
09C5H	Bump the value of the memory pointer in register pair HL.
09C6H	Load register C with the MSB of the single precision value in REG1 at the location of the memory pointer in register pair HL.
09C7H	Bump the value of the memory pointer in register pair HL.
09C8H	Load register B with the exponent of the single precision value in REG1 at the location of the memory pointer in register pair HL.
09C9H	Bump the value of the memory pointer in register pair HL.

09CAH		Return.
09CBH - 09D1H		**SINGLE PRECISION MATH ROUTINE**
09CBH - 09CDH		Load register pair DE with the starting address for a single precision value in REG1.
09CEH - 09CFH		Load register B with the number of bytes to be moved for a single precision value.
09D0H - 09D1H		Jump.
09D2H - 09DEH		**MOVE VALUE POINTED TO BY HL TO THE LOCATION POINTED TO BY DE**
09D2H		Exchange the value in register pair HL with the value in register pair DE.
09D3H - 09D5H		Load register A with the current value of the number type flag.
09D6H		Load register B with the number of bytes to be moved in register A.
09D7H		Load register A with the value at the location of the memory pointer in register pair DE.
09D8H		Save the value in register A at the location of the memory pointer in register pair HL.
09D9H		Bump the value of the memory pointer in register pair DE.
09DAH		Bump the value of the memory pointer in register pair HL.
09DBH		Decrement the value of the byte counter in register B.
09DCH - 09DDH		Loop till all of the bytes have been moved.
09DEH		Return.
09DFH - 09F3H		**SINGLE PRECISION MATH ROUTINE**
09DFH - 09E1H		Load register pair HL with the address of the MSB of the value in REG1.
09E2H		Load register A with the MSB of the value in REG1 at the location of the memory pointer in register pair HL.
09E3H		Move the value of the sign bit in register A into the Carry flag.
09E4H		Set the Carry flag.

09E5H	Turn of the sign bit in register A by moving the value of the Carry flag into register A and moving the previous value of the sign bit from bit 0 of register A into the Carry flag.
09E6H	Save the adjusted MSB in register A in REG1 at the location of the memory pointer in register pair HL.
09E7H	Invert the value of the sign bit in the Carry flag.
09E8H	Move the inverted sign bit from the Carry flag into register A.
09E9H	Bump the value of the memory pointer in register pair HL.
09EAH	Bump the value of the memory pointer in register pair HL.
09EBH	Save the value in register A at the location of the memory pointer in register pair HL.
09ECH	Load register A with the MSB of the single precision value in register C.
09EDH	Move the value of the sign bit in register A into the Carry flag.
09EEH	Set the Carry flag.
09EFH	Turn on the sign bit in register A by moving the value of the Carry flag into register A and moving the previous value of the sign bit from bit 0 of register A into the Carry flag.
09F0H	Load register C with the adjusted MSB in register A.
09F1H	Move the value of the sign bit from the Carry flag into register A.
09F2H	Combine the value of the sign bit for the single precision value in register A with the value of the sign bit for the single precision value at the location of the memory pointer in register pair HL.
09F3H	Return.
09F4H - 09FBH	**LEVEL II BASIC MATH ROUTINE**
09F4H - 09F6H	Load register pair HL with the starting address of REG2.
09F7H - 09F9H	Load register pair DE with the return address.
09FAH - 09FBH	Jump.

09FCH - 0A0BH	**LEVEL II BASIC MATH ROUTINE**
09FCH - 09FEH	Load register pair HL with the starting address of REG2.
09FFH - 0A01H	Load register pair DE with the return address.
0A02H	Save the return address in register pair DE on the stack.
0A03H - 0A05H	Load register pair HL with the starting address for a single precision value in REG1.
0A06H	Go check the current value of the number type flag.
0A07H	Return if the current value in REG1 isn't double precision.
0A08H - 0A0AH	Load register pair DE with the starting address of REG1.
0A0BH	Return.
0A0CH - 0A25H	**SINGLE PRECISION COMPARE**
0A0CH	Load register A with the value of the exponent in register B.
0A0DH	Check to see if the exponent in register A is equal to zero.
0A0EH - 0A10H	Jump if the exponent in register A is equal to zero.
0A11H - 0A13H	Load register pair HL with the return address.
0A14H	Save the return address in register pair HL on the stack.
0A15H - 0A17H	Go check the signs of the single precision numbers.
0A18H	Load register A with the MSB of the single precision value in register C.
0A19H	Return if the signs of the single precision values aren't equal.
0A1AH - 0A1CH	Load register pair HL with the address of the MSB in REG1.
0A1DH	Combine the MSB in REG1 at the location of the memory pointer in register pair HL with the MSB in register A.
0A1EH	Load register A with the MSB in register C.
0A1FH	Return if the MSBs don't match.

0A20H - 0A22H	Go compare the single precision values.
0A23H	Move the value of the Carry flag from the comparison into register A.
0A24H	Combine the value of the MSB of the single precision value in register C with the value in register A.
0A25H	Return.
0A26H - 0A38H	**SINGLE PRECISION COMPARISON ROUTINE**
0A26H	Bump the value of the memory pointer in register pair HL so that it points to the exponent in REG1.
0A27H	Load register A with the value of the exponent for the single precision value in register B.
0A28H	Check to see if the exponent for the single precision value in REG1 at the location of the memory pointer in register pair HL is the same as the value of the exponent for the single precision value in register A.
0A29H	Return if the value of the exponent in REG1 isn't the same as the value of the exponent in register A.
0A2AH	Decrement the value of the memory pointer in register pair HL.
0A2BH	Load register A with the MSB for the single precision value in register C.
0A2CH	Check to see if the MSB for the single precision value in REG1 at the location of the memory pointer in register pair HL is the same as the value of the MSB for the single precision value in register A.
0A2DH	Return if the value of the MSB in REG1 isn't the same as the value of the MSB in register A.
0A2EH	Decrement the value of the memory pointer in register pair HL.
0A2FH	Load register A with the NMSB of the single precision value in register D.
0A30H	Check to see if the NMSB for the single precision value in REG1 at the location of the memory pointer in register pair HL is the same as the value of the NMSB for the single precision value in register A.
0A31H	Return if the value of the NMSB in REG1 isn't the same as the value of the NMSB in register A.
0A32H	Decrement the value of the memory pointer in register pair HL.

0A33H	Load register A with the LSB of the single precision value in register E.
0A34H	Check to see if the LSB for the single precision value in REG1 at the location of the memory pointer in register pair HL is the same as the value of the LSB for the single precision value in register A.
0A35H	Return if the value of the LSB in REG1 isn't the same as the value of the LSB in register A.
0A36H	Get the value from the stack and put it in register pair HL.
0A37H	Get the value from the stack and put it in register pair HL.
0A38H	Return.
0A39H - 0A48H	**INTEGER COMPARE**
0A39H	Load register A with the MSB of the integer value in register D.
0A3AH	Check to see if the sign bit for the MSB of the integer value in register H is the same as the sign bit for the MSB for the integer value in register A.
0A3BH	Load register A with the MSB of the integer value in register H.
0A3CH - 0A3EH	Jump if the sign bits for the integer values aren't equal.
0A3FH	Check to see if the MSB for the integer value in register D is the same as the MSB for the integer value in register A.
0A40H - 0A42H	Jump if the MSB for the integer value in register D isn't the same as the MSB for the integer value in register A.
0A43H	Load register A with the LSB of the integer value in register L.
0A44H	Check to see if the LSB for the integer value in register E is the same as the LSB for the integer value in register A.
0A45H - 0A47H	Jump if the LSB for the integer value in register E isn't the same as the LSB for the integer value in register A.
0A48H	Return.
0A49H - 0A77H	**DOUBLE PRECISION COMPARE**

0A49H - 0A4BH	Load register pair HL with the starting address of REG2.
0A4CH - 0A4EH	Go move the double precision value pointed to by register pair DE to REG2.
0A4FH - 0A51H	Load register pair DE with the address of the exponent in REG2.
0A52H	Load register A with the exponent for the double precision value in REG2 at the location of the memory pointer in register pair DE.
0A53H	Check to see if the double precision value in REG2 is equal to zero.
0A54H - 0A56H	Jump if the double precision value in REG2 is equal to zero.
0A57H - 0A59H	Load register pair HL with the return address.
0A5AH	Save the return address in register pair HL on the stack.
0A5BH - 0A5DH	Go check to see if the double precision value in REG1 is equal to zero.
0A5EH	Decrement the value of the memory pointer in register pair DE.
0A5FH	Load register A with the MSB of the double precision value in REG2 at the location of the memory pointer in register pair DE.
0A60H	Load register C with the MSB of the double precision value in REG2 in register A.
0A61H	Return if the double precision value in REG1 is equal to zero.
0A62H - 0A64H	Load register pair HL with the address of the MSB of the double precision value in REG1.
0A65H	Check to see if the sign bit for the MSB of the double precision value in REG1 at the location of the memory pointer in register pair HL is the same as the sign bit for the MSB of the double precision value in REG2 in register A.
0A66H	Load register A with the MSB of the double precision value in REG2 in register C.
0A67H	Return if the sign bits for the double precision values in REG1 and REG2 aren't the same.
0A68H	Bump the value of the memory pointer in register pair DE.

0A69H		Bump the value of the memory pointer in register pair HL.
0A6AH	- 0A6BH	Load register B with the number of bytes to be compared.
0A6CH		Load register A with the value at the location of the memory pointer in register pair DE.
0A6DH		Check to see if the value in register A is the same as the value at the location of the memory pointer in register pair HL.
0A6EH	- 0A70H	Jump if the value in register A isn't the same as the value at the location of the memory pointer in register pair HL.
0A71H		Decrement the value of the memory pointer in register pair DE.
0A72H		Decrement the value of the memory pointer in register pair HL.
0A73H		Decrement the number of bytes remaining to be compared in register B.
0A74H	- 0A75H	Loop till all of the bytes have been compared.
0A76H		Get the value from the stack and put it in register pair BC.
0A77H		Return.
0A78H	- 0A7EH	**DOUBLE PRECISION COMPARE**
0A78H	- 0A7AH	Go compare the double precision value in REG2 to the double precision value in REG1.
0A7BH	- 0A7DH	Jump if the double precision value in REG1 and the double precision value in REG2 aren't the same.
0A7EH		Return.
0A7FH	- 0AB0H	**LEVEL II BASIC CINT ROUTINE**
0A7FH		Go check the current value of the number type flag.
0A80H	- 0A82H	Load register pair HL with the integer value in REG1.
0A83H		Return if the current value in REG1 is an integer.
0A84H	- 0A86H	Go to the Level II BASIC error routine and display a TM ERROR message if the current value in REG1 is a string.

0A87H - 0A89H	Go convert the double precision value in REG1 to single precision.
0A8AH - 0A8CH	Load register pair HL with the return address.
0A8DH	Save the return address in register pair HL on the stack.
0A8EH - 0A90H	Load register A with the exponent for the single precision value in REG1.
0A91H - 0A92H	Check to see if the exponent for the single precision value in REG1 in register A indicates more than 16 bits of precision.
0A93H - 0A94H	Jump if the exponent for the single precision value in REG1 in register A indicates more than 16 bits of precision.
0A95H - 0A97H	Go convert the single precision value in REG1 to an integer and return with the integer value in register pair DE.
0A98H	Load register pair HL with the integer value in register pair DE.
0A99H	Get the value from the stack and put it in register pair DE.
0A9AH - 0A9CH	Save the integer value in register pair HL as the current value in REG1.
0A9DH - 0A9EH	Load register A with an integer number type flag.
0A9FH - 0AA1H	Save the integer number type flag in register A as the current value of the number type flag.
0AA2H	Return.
0AA3H - 0AA5H	Load register pair BC with the exponent and the MSB of a single precision value.
0AA6H - 0AA8H	Load register pair DE with the NMSB and the LSB of a single precision value. Register pairs BC and DE now hold a single precision value equal to −32768.
0AA9H - 0AABH	Go check to see if the single precision value in REG1 is equal to −32768.
0AACH	Go to the Level II BASIC error routine and display an OV ERROR message if the value in REG1 isn't equal to −32768.
0AADH	Load register H with the MSB of the single precision value in register C.

0AAEH	Load register L with the NMSB of the single precision value in register D.
0AAFH - 0AB0H	Jump.
0AB1H - 0ACBH	**LEVEL II BASIC CSNG ROUTINE**
0AB1H	Go check the current value of the number type flag.
0AB2H	Return if the current value in REG1 is single precision.
0AB3H - 0AB5H	Jump if the current value in REG1 is an integer.
0AB6H - 0AB8H	Go to the Level II BASIC error routine and display a TM ERROR message if the current value in REG1 is a string.
0AB9H - 0ABBH	Go move the most significant four bytes of the double precision value in REG1 into register pairs BC and DE.
0ABCH - 0ABEH	Go set the current number type flag to single precision.
0ABFH	Load register A with the exponent of the double precision value in register B.
0AC0H	Check to see if the double precision value in REG1 is equal to zero.
0AC1H	Return if the double precision value in REG1 is equal to zero.
0AC2H - 0AC4H	Go turn on the most significant bit of the single precision value in REG1.
0AC5H - 0AC7H	Load register pair HL with the address of the most significant byte chopped off the double precision value.
0AC8H	Loaded register B with the most significant byte chopped off the double precision value at the location of the memory pointer in register pair HL.
0AC9H - 0ACBH	Jump.
0ACCH - 0ADAH	**LEVEL II BASIC MATH ROUTINE**
0ACCH - 0ACEH	Load register pair HL with the integer value at the location of an integer storage in REG1.
0ACFH - 0AD1H	Go set the current number type flag to single precision.
0AD2H	Load register A with the MSB of the integer value in register H.

0AD3H	Load register D with the LSB of the integer value in register L.
0AD4H - 0AD5H	Zero register E.
0AD6H - 0AD7H	Load register B with an exponent.
0AD8H - 0ADAH	Jump.
0ADBH - 0AEDH	**LEVEL II BASIC CDBL ROUTINE**
0ADBH	Go check the current value of the number type flag.
0ADCH	Return if the current value in REG1 is double precision.
0ADDH - 0ADFH	Go to the Level II BASIC error routine and display a TM ERROR message if the current value in REG1 is a string.
0AE0H - 0AE2H	Go convert an integer to single precision.
0AE3H - 0AE5H	Load register pair HL with zero.
0AE6H - 0AE8H	Save the value in register pair HL as the first and second bytes of REG1.
0AE9H - 0AEBH	Save the value in register pair HL as the third and fourth bytes of REG1.
0AECH - 0AEDH	Load register A with a double precision number type flag.
0AEEH - 0AF3H	**LEVEL II BASIC MATH ROUTINE**
0AEFH - 0AF0H	Load register A with a single precision number type flag.
0AF1H - 0AF3H	Save the value in register A as the current number type flag.
0AF4H - 0AFAH	**LEVEL II BASIC MATH ROUTINE**
0AF4H	Go check the current value of the number type flag.
0AF5H	Return if the current value in REG1 is a string.
0AF6H - 0AF7H	Load register E with a TM ERROR code.
0AF8H - 0AFAH	Go to the LEVEL II BASIC error routine and display a TM ERROR message if the current value in REG1 isn't a string.
0AFBH - 0B1EH	**LEVEL II BASIC MATH ROUTINE**
0AFBH	Load register B with the exponent of the single precision number in register A.

0AFCH	Load register C with the exponent of the single precision number in register A.
0AFDH	Load register D with the exponent of the single precision number in register A.
0AFEH	Load register E with the exponent of the single precision number in register A.
0AFFH	Check to see if the single precision number in REG1 is equal to zero.
0B00H	Return if the single precision value in REG1 is equal to zero.
0B01H	Save the value in register pair HL on the stack.
0B02H - 0B04H	Go move the single precision value in REG1 into register pairs BC and DE.
0B05H - 0B07H	Go turn on the sign bit of the single precision value in register pairs BC and DE.
0B08H	Set the sign bit according to the sign of the value at the location of the memory pointer in register pair HL.
0B09H	Load register H with the value of the sign in register A.
0B0AH - 0B0CH	Go decrement the single precision value in register pairs BC and DE if the sign of the value is negative.
0B0DH - 0B0EH	Load register A with the maximum exponent.
0B0FH	Figure the number of bits to be shifted by subtracting the exponent in register B from the exponent in register A.
0B10H - 0B12H	Go shift the single precision value in register pairs BC and DE.
0B13H	Load register A with the value of the sign in register H.
0B14H	Put the sign bit into the Carry flag.
0B15H - 0B17H	Go bump the value in register pairs BC and DE if the original number was negative.
0B18H - 0B19H	Load register B with an exponent.
0B1AH - 0B1CH	Go make the integer negative if the original number was negative. Return with the integer value in register pair DE.

0B1DH		Get the value from the stack and put it in register pair HL.
0B1EH		Return.
0B1FH - 0B25H		**LEVEL II BASIC MATH ROUTINE**
0B1FH		Decrement the NMSB and the LSB of the single precision value in register pair DE.
0B20H		Load register A with the value of the NMSB for the single precision value in register D.
0B21H		Combine the LSB of the single precision value in register E with the NMSB of the single precision value in register A.
0B22H		Bump the combined value in register A.
0B23H		Return if the NMSB and the LSB of the single precision value in register pair DE isn't equal to FFFFH.
0B24H		Decrement the value of the exponent and the MSB of the single precision value in register pair BC.
0B25H		Return.
0B26H - 0B58H		**LEVEL II BASIC FIX ROUTINE**
0B26H		Go check the current value of the number type flag.
0B27H		Return if the current value in REG1 is an integer.
0B28H - 0B2AH		Go check the sign of the current value in REG1.
0B2BH - 0B2DH		Jump if the current value in REG1 is positive.
0B2EH - 0B30H		Go convert the current value in REG1 to positive.
0B31H - 0B33H		Go get the integer part of the current value in REG1.
0B34H - 0B36H		Jump.
0B37H		Go check the current value of the number type flag.
0B38H		Return if the current value in REG1 is an integer.
0B39H - 0B3AH		Jump if the current value in REG1 is double precision.
0B3BH - 0B3CH		Go to the Level II BASIC error routine and display a TM ERROR message if the current value in REG1 is a string.
0B3DH - 0B3FH		Go convert the single precision value in REG1 to an integer.

0B40H - 0B42H	Load register pair HL with the address of the exponent in REG1.
0B43H	Load register A with the value of the exponent in REG1 at the location of the memory pointer in register pair HL.
0B44H - 0B45H	Check to see if there is more than 24 bits of precision used by the current value in REG1.
0B46H - 0B48H	Load register A with the LSB of the single precision number in REG1.
0B49H	Return if more than 24 bits of precision is used by the single precision value in REG1.
0B4AH	Load register A with the exponent of the single precision number in REG1.
0B4BH - 0B4DH	Go convert the single precision number in REG1 to an integer.
0B4EH - 0B4FH	Save an exponent in REG1.
0B50H	Load register A with the LSB of the integer value in register E.
0B51H	Save the LSB of the integer value in register A on the stack.
0B52H	Load register A with the value in register C.
0B53H	Move the sign bit in register A into the Carry flag.
0B54H - 0B56H	Go make the number in REG1 single precision.
0B57H	Get the LSB of the single precision value from the stack and put it in register A.
0B58H	Return.
0B59H - 0B9DH	**LEVEL II BASIC MATH ROUTINE**
0B59H - 0B5BH	Load register pair HL with the address of the exponent in REG1.
0B5CH	Load register A with the value of the exponent for the double precision number in REG1 at the location of the memory pointer in register pair HL.
0B5DH - 0B5EH	Check to see if the double precision number in REG1 uses more or less than 16 bits of precision.
0B5FH - 0B61H	Jump if the double precision value in REG1 uses less than 16 bits of precision.

0B62H - 0B63H	Jump if the double precision number in REG1 uses more than 16 bits of precision.
0B64H	Load register C with the exponent for the double precision number in register A.
0B65H	Decrement the value of the memory pointer in register pair HL.
0B66H	Load register A with the MSB of the double precision value in REG1 at the location of the memory pointer in register pair HL.
0B67H - 0B68H	Complement the value of the sign bit in register A.
0B69H - 0B6AH	Load register B with the number of bytes to be checked.
0B6BH	Decrement the value of the memory pointer in register pair HL.
0B6CH	Combine the value at the location of the memory pointer in register HL with the value in register A.
0B6DH	Decrement the byte counter in register B.
0B6EH - 0B6FH	Loop till all of the bytes have been checked.
0B70H	Check to see if the double precision value in REG1 is equal to a −32768.
0B71H - 0B73H	Load register pair HL with a negative zero.
0B74H - 0B76H	Jump if the double precision value in REG1 is a −32768.
0B77H	Load register A with the exponent for the double precision value in register C.
0B78H - 0B79H	Check to see if more than 56 bits of precision has been used for the double precision value in REG1.
0B7AH	Return if more than 56 bits of precision have been used for the double precision value in REG1.
0B7BH	Save the exponent in register A on the stack.
0B7CH - 0B7EH	Go move the most significant four bytes of the double precision value in REG1 into register pairs BC and DE.
0B7FH - 0B81H	Go turn on the sign bit and return with the value of the sign.
0B82H	Combine the value at the location of the memory

	pointer in register pair HL with the result of the sign test in register A.
0B83H	Decrement the value of the memory pointer in register pair HL.
0B84H - 0B85H	Save an exponent at the location of the memory pointer in register HL.
0B86H	Save the value of the sign test in register A on the stack.
0B87H - 0B89H	Go adjust the number in REG1 if it's negative.
0B8AH - 0B8GH	Load register pair HL with the address of the MSB in REG1.
0B8DH - 0B8EH	Load register A with the maximum value of an exponent.
0B8FH	Subtract the value of the exponent at the location of the memory pointer in register pair HL from the value in register A.
0B90H - 0B92H	Go adjust the value in REG1.
0B93H	Get the value of the sign test from the stack and put it in register A.
0B94H - 0B96H	Go adjust the value in REG1 if it's negative.
0B97H	Zero register A.
0B98H - 0B9AH	Zero memory location 411CH.
0B9BH	Get the value of the original exponent test from the stack and put it in register pair AF.
0B9CH	Return if more than 56 bits are used by the double precision number.
0B9DH - 0B9FH	Jump.
0BA0H - 0BA9H	**LEVEL II BASIC MATH ROUTINE**
0BA0H - 0BA2H	Load register pair HL with the starting address of REG1.
0BA3H	Load register A with the value at the location of the memory pointer in register pair HL.
0BA4H	Decrement the value at the location of the memory pointer in register pair HL.
0BA5H	Check to see if the value in register A is equal to zero.

0BA6H	Decrement the value of the memory pointer in register pair HL.
0BA7H - 0BA8H	Loop till the value at the location of the memory pointer in register pair HL is equal to a nonzero value.
0BA9H	Return.
0BAAH - 0BC6H	**LEVEL II BASIC MATH ROUTINE**
0BAAH	Save the value in register pair HL on the stack.
0BABH - 0BADH	Load register pair HL with zero.
0BAEH	Load register A with the MSB of the integer value in register B.
0BAFH	Combine the LSB of the integer value in register C with the MSB of the integer value in register A.
0BB0H - 0BB1H	Jump if the integer value in register pair BC is equal to zero.
0BB2H - 0BB3H	Load register A with the counter value.
0BB4H	Multiply the result in register pair HL by two.
0BB5H - 0BB7H	Go to the Level II BASIC error routine and display a BS ERROR message if the result in register pair HL has overflowed.
0BB8H	Exchange the integer value in register pair DE with the result in register pair HL.
0BB9H	Multiply the integer value in register pair HL by two.
0BBAH	Exchange the result in register pair DE with the integer value in register pair HL.
0BBBH - 0BBCH	Jump if the most significant bit in the integer value in register DE wasn't set.
0BBDH	Add the integer value in register pair BC to the result in register pair HL.
0BBEH - 0BC0H	Go to the Level II BASIC error routine and display a BS ERROR message if the result in register pair HL has overflowed.
0BC1H	Decrement the counter in register A.
0BC2H - 0BC3H	Loop till the multiplication has been completed.
0BC4H	Load register pair DE with the result in register pair HL.

0BC5H	Get the value from the stack and put it in register pair HL.
0BC6H	Return.
0BC7H - 0BD1H	**INTEGER SUBTRACTION**
0BC7H	Load register A with the MSB of the integer value in register H.
0BC8H	Put the value of the sign bit into the Carry flag.
0BC9H	Adjust register A according to the value of the sign bit.
0BCAH	Load register B with the result of the sign test.
0BCBH - 0BCDH	Go complement the value in register pair HL.
0BCEH	Load register A with zero.
0BCFH	Adjust the value of the sign test.
0BD0H - 0BD1H	Jump.
0BD2H - 0BF1H	**INTEGER ADDITION**
0BD2H	Load register A with the MSB of the integer value in register pair HL.
0BD3H	Put the value of the sign bit into the Carry flag.
0BD4H	Adjust register A according to the value of the sign bit.
0BD5H	Load register B with the result of the sign test.
0BD6H	Save the integer value in register pair HL on the stack.
0BD7H	Load register A with the MSB of the integer value in register pair DE.
0BD8H	Put the value of the sign bit into the Carry flag.
0BD9H	Adjust register A according to the value of the sign bit.
0BDAH	Add the integer value in register pair DE to the integer value in register pair HL.
0BDBH	Add the Carry flag and the value from the sign test for the integer value in register pair HL in register B to the value of the sign test for the integer value in register pair DE in register A.

0BDCH	Put the value of the Carry flag in register A.
0BDDH	Combine the value of the sign bit for the result in register H with the value in register A.
0BDEH - 0BE0H	Jump unless overflow has occurred.
0BE1H	Save the value of the sign test in register B on the stack.
0BE2H	Load register pair HL with the integer value in register pair DE.
0BE3H - 0BE5H	Go convert the register value in register pair HL to single precision and return with the result in REG1.
0BE6H	Get the value of the sign test from the stack and put it in register A.
0BE7H	Get the integer value from the stack and put it in register pair HL.
0BE8H - 0BEAH	Go move the single precision value in REG1 on to the stack.
0BEBH	Load register pair DE with the integer value in register pair HL.
0BECH - 0BEEH	Go convert the integer value in register pair DE to single precision and return with the result in REG1.
0BEFH - 0BF1H	Go perform single precision addition.
0BF2H - 0C1EH	**INTEGER MULTIPLICATION**
0BF2H	Load register A with the MSB of the integer value in register H.
0BF3H	Combine the LSB of the integer value in register L with the MSB of the integer value in register A.
0BF4H - 0BF6H	Jump if the integer value in register pair HL is equal to zero.
0BF7H	Save the integer value in register pair HL on the stack.
0BF8H	Save the integer value in register pair DE on the stack.
0BF9H - 0BFBH	Go convert any negative integer values to positive and return with register B set according to the value of the sign bits.
0BFCH	Save the value of the sign bit test in register B on the stack.

0BFDH	Load register B with the MSB of the integer value in register H.
0BFEH	Load register C with the LSB of the integer value in register L.
0BFFH - 0C01H	Load register pair HL with zero.
0C02H - 0C03H	Load register A with the counter value.
0C04H	Multiply the result in register pair HL by two.
0C05H - 0C06H	Jump if the result in register pair HL has overflowed.
0C07H	Exchange the integer value in register pair DE with the integer result in register pair HL.
0C08H	Multiply the integer value in register pair HL by two.
0C09H	Exchange the integer result in register pair DE with the integer value in register pair HL.
0C0AH - 0C0BH	Jump if the most significant bit of the integer register pair DE wasn't set.
0C0CH	Add the integer value in register pair BC to the integer result in register pair HL.
0C0DH - 0C0FH	Jump if the result in register pair HL has overflowed.
0C10H	Decrement the value of the counter in register A.
0C11H - 0C12H	Loop till the multiplication has been completed.
0C13H	Get the value of the sign test from the stack and put it in register B.
0C14H	Get the integer value from the stack and put it in register pair DE.
0C15H	Load register A with the MSB of the result in register H.
0C16H	Check to see if the sign bit in the MSB of the integer result in register A is set.
0C17H - 0C19H	Jump if the integer result in register pair HL is negative.
0C1AH	Get the integer value from the stack and put it in register pair DE.
0C1BH	Load register A with the value of the sign test in register B.

0C1CH - 0C1EH		Jump.
0C1FH - 0C34H		**LEVEL II BASIC MATH ROUTINE**
0C1FH - 0C20H		Clear the sign bit for the MSB of the integer value in register A.
0C21H		Combine the value of the LSB for the integer value in register L with the adjusted MSB of the integer value in register A.
0C22H - 0C23H		Jump if the result is equal to zero.
0C24H		Load register pair HL with the integer value in register pair DE.
0C26H		Get the value of the sign test from the stack and put it in register B.
0C27H		Get the integer value from the stack and put it in register pair HL.
0C28H - 0C2AH		Go convert the integer value in register pair HL to single precision and return with the result in REG1.
0C2BH		Get the integer value from the stack and put it in register pair HL.
0C2CH - 0C2EH		Go move the single precision value from REG1 to the stack.
0C2FH - 0C31H		Go convert the integer value in register pair HL to single precision and return with the result in REG1.
0C32H		Get the exponent and the MSB of the single precision value from the stack and put it in register pair BC.
0C33H		Get the NMSB and the LSB of the single precision value from the stack and put it in register pair DE.
0C34H - 0C36H		Go do single precision multiplication.
0C37H - 0C44H		**LEVEL II BASIC MATH ROUTINE**
0C37H		Load register A with the result of the sign test in register B.
0C38H		Check to see if the result is supposed to be negative.
0C39H		Get the value from the stack and put it in register pair BC.
0C3AH - 0C3CH		Jump if the result is supposed to be negative.

0C3DH		Save the integer value in register pair DE on the stack.
0C3EH	- 0C40H	Go convert the integer result in register pair HL to single precision and return with the result in REG1.
0C41H		Get the integer value from the stack and put it in register pair DE.
0C42H	- 0C44H	Jump.
0C45H	- 0C5AH	**LEVEL II BASIC MATH ROUTINE**
0C45H		Load register A with the MSB of the integer value in register H.
0C46H		Combine the MSB of the integer value in register D with the MSB of the integer value in register A.
0C47H		Save the result of the combined signs in register A into register B.
0C48H	- 0C4AH	Go convert a negative integer value in register pair HL to positive.
0C4BH		Exchange the integer value in register pair DE with the integer value in register pair HL.
0C4CH		Load register A with the MSB of the integer value in register H.
0C4DH		Check the value of the sign for the MSB of the integer value in register A.
0C4EH	- 0C50H	Jump if the integer value in register pair HL is positive.
0C51H		Zero register A.
0C52H		Load register C with the value in register A.
0C53H		Subtract the LSB of the integer value in register L from the value in register A.
0C54H		Save the adjusted value in register A in register L.
0C55H		Load register A with the value in register C.
0C56H		Subtract the MSB of the integer value in register H from the value in register A.
0C57H		Save the adjusted value in register A into register H.
0C58H	- 0C5AH	Jump.
0C5BH	- 0C6FH	**LEVEL II BASIC MATH ROUTINE**

0C5BH - 0C5DH	Load register pair HL with the integer value in REG1.
0C5EH - 0C60H	Go convert the integer value in register pair HL to positive if it's negative.
0C61H	Load register A with the MSB of the integer value in register H.
0C62H - 0C63H	Invert the value of the sign bit in register A.
0C64H	Combine the LSB of the integer value in register L with the adjusted MSB of the integer value in register A.
0C65H	Return if the integer value in REG1 isn't equal to −32768.
0C66H	Load register pair DE with the integer value in register pair HL.
0C67H - 0C69H	Go set the number type flag to single precision.
0C6AH	Zero register A.
0C6BH - 0C6CH	Load register B with an exponent.
0C6DH - 0C6FH	Jump.
0C70H - 0C76H	**DOUBLE PRECISION SUBTRACTION**
0C70H - 0C72H	Load register pair HL with the address of the MSB in REG2.
0C73H	Load register A with the MSB of the double precision value in REG2 at the location of the memory pointer in register pair HL.
0C74H - 0C75H	Invert the value of the sign bit for the MSB of the double precision value in register A.
0C76H	Save the adjusted MSB of the double precision value in register A in REG2 at the location of the memory pointer in register pair HL.
0C77H - 0CCEH	**DOUBLE PRECISION ADDITION**
0C77H - 0C79H	Load register pair HL with the address of the exponent in REG2.
0C7AH	Load register A with the exponent of the double precision value in REG2 at the location of the memory pointer in register pair HL.
0C7BH	Check to see if the double precision value in REG2 is equal to zero.

0C7CH	Return if the double precision value in REG2 is equal to zero.
0C7DH	Load register B with the value of the exponent for the double precision value in register A.
0C7EH	Decrement the value of the memory pointer in register pair HL.
0C7FH	Load register C with the value of the MSB of the double precision value in REG2 at the location of the memory pointer in register pair HL.
0C80H - 0C82H	Load register pair DE with the address of the exponent in REG1.
0C83H	Load register A with the value of the exponent of the double precision value in REG1 at the location of the memory pointer in register pair DE.
0C84H	Check to see if the double precision value in REG1 is equal to zero.
0C85H - 0C87H	Jump if the double precision value in REG1 is equal to zero.
0C88H	Subtract the value of the exponent for the double precision value in REG2 in register B from the value of the exponent for the double precision value in REG1 in register A.
0C89H - 0C8AH	Jump if the double precision value in REG1 is greater than or equal to the double precision value in REG2.
0C8BH	Complement the difference between the two exponents in register A.
0C8CH	Bump the value of the difference for the exponents in register A so that register A will hold the positive difference.
0C8DH	Save the difference for the exponents in register A on the stack.
0C8EH - 0C8FH	Load register C with the counter value.
0C90H	Bump the value of the memory pointer in register pair HL so that it will be pointing to the exponent of the double precision value in REG2.
0C91H	Save the value of the memory pointer in register pair HL on the stack.
0C92H	Load register A with the value at the location of the memory pointer in register pair DE.

0C93H	Load register B with the value at the location of the memory pointer in register pair HL.
0C94H	Save the value in register A at the location of the memory pointer in register pair HL.
0C95H	Load register A with the value in register B.
0C96H	Save the value in register A at the location of the memory pointer in register pair DE.
0C97H	Decrement the value of the memory pointer in register pair DE.
0C98H	Decrement the value of the memory pointer in register pair HL.
0C99H	Decrement the value of the counter in register C.
0C9AH - 0C9BH	Loop till the double precision values in REG1 and REG2 have been exchanged.
0C9CH	Get the value of the memory pointer from the stack and put it in register pair HL.
0C9DH	Load register B with the value of the exponent for the double precision value in REG2 at the location of the memory pointer in register pair HL.
0C9EH	Decrement the value of the memory pointer in register pair HL.
0C9FH	Load register C with the value of the MSB for the double precision value in REG2 at the location of the memory pointer in register pair HL.
0CA0H	Get the difference for the exponents from the stack and put it in register A.
0CA1H - 0CA2H	Check to see if the difference between the two exponents is greater than 56 bits.
0CA3H	Return if the difference between the two exponents is greater than 56 bits.
0CA4H	Save the difference for the exponents in register A on the stack.
0CA5H - 0CA7H	Go turn on the sign bits for the double precision numbers.
0CA8H	Bump the value of the memory pointer in register pair HL.
0CA9H - 0CAAH	Zero the location of the memory pointer in register pair HL.

0CABH	Save the result of the sign test in register A in register B.
0CACH	Get the difference for the exponents from the stack and put it in register A.
0CADH - 0CAFH	Load register pair HL with the address of the MSB in REG2.
0CB0H - 0CB2H	Go shift the double precision value in REG2 till it lines up with the double precision value in REG1.
0CB3H - 0CB5H	Get the bits shifted out of the double precision value in REG2 and put it in register A.
0CB6H - 0CB8H	Save the value in register A.
0CB9H	Load register A with the value of the sign test in register B.
0CBAH	Check to see if the signs are equal.
0CBBH - 0CBDH	Go subtract the values if the signs aren't equal.
0CBEH - 0CC0H	Go add the values.
0CC1H - 0CC3H	Jump if the Carry flag isn't set.
0CC4H	Load register pair HL with the value of the memory pointer in register pair DE.
0CC5H	Bump the value of the exponent at the location of the memory pointer in register pair HL.
0CC6H - 0CC8H	Go to the Level II BASIC error routine and display an OV ERROR message if the exponent for the double precision result in REG1 is too large.
0CC9H - 0CCBH	Go adjust the double precision result in REG1.
0CCCH - 0CCEH	Jump.
0CCFH - 0D1FH	**DOUBLE PRECISION MATH ROUTINE**
0CCFH - 0CD1H	Go subtract the double precision values.
0CD2H - 0CD4H	Load register pair HL with the address of the sign for the result.
0CD5H - 0CD7H	Go complement the result in REG1 if the Carry flag is set.
0CD8H	Zero register A.
0CD9H	Load register B with the value in register A.

0CDAH - 0CDCH	Load register A with the value of the MSB of the double precision result in REG1.
0CDDH	Check to see if the MSB of the double precision result is equal to zero.
0CDEH - 0CDFH	Jump if the MSB of the double precision value is nonzero.
0CE0H - 0CE2H	Load register pair HL with the starting address of REG1 minus one.
0CE3H - 0CE4H	Load register C with the number of bytes to be shifted.
0CE5H	Load register D with the value at the location of the memory pointer in register pair HL.
0CE6H	Save the value in register A at the location of the memory pointer in register pair HL.
0CE7H	Load register A with the value in register D.
0CE8H	Bump the value of the memory pointer in register pair HL.
0CE9H	Decrement the number of bytes to be shifted in register C.
0CEAH - 0CEBH	Loop till all of the bytes in the double precision value have been shifted.
0CECH	Load register A with the number of bits shifted in register B.
0CEDH - 0CEEH	Subtract the number of bits just shifted from the shift counter in register A.
0CEFH - 0CF0H	Check to see if the whole of the double precision value has been shifted.
0CF1H - 0CF2H	Jump if the whole of the double precision value hasn't been shifted.
0CF3H - 0CF5H	Go make the double precision value equal to zero if the whole of the double precision value has been shifted.
0CF6H	Decrement the shift counter in register B.
0CF7H - 0CF9H	Load register pair HL with the starting address of REG1 minus one.
0CFAH - 0CFCH	Go shift the double precision value in REG1 once.
0CFDH	Check to see if the sign bit of the MSB of the double precision value in register A is set.

0CFEH - 0D00H	Loop till the sign bit is set.
0D01H	Load register A with the value of the shift counter in register B.
0D02H	Check to see if the shift counter in register A is equal to zero.
0D03H - 0D04H	Jump if the value in REG1 wasn't shifted at all.
0D05H - 0D07H	Load register pair HL with the address of the exponent in REG1.
0D08H	Add the value of the exponent for the double precision value in REG1 at the location of the memory pointer in register pair HL to the value of the shift counter in register A.
0D09H	Save the adjusted exponent for the double precision value in register A at the location of the memory pointer in register pair HL.
0D0AH - 0D0CH	Jump if overflow didn't occur.
0D0DH	Return if the double precision value is equal to zero.
0D0EH - 0D10H	Load register A with the value of the rounding byte at the location of the starting address of REG1 minus one.
0D11H	Check to see if there is a bit to be shifted into the double precision value in REG1.
0D12H - 0D14H	Go move the bit into the double precision value if necessary.
0D15H - 0D17H	Load register pair HL with the address of the sign for the result.
0D18H	Load register A with the value of the sign for the result at the location of the memory pointer in register pair HL.
0D19H - 0D1AH	Mask the value of the sign for the result in register A.
0D1BH	Decrement the value of the memory pointer in register pair HL so that it points to the exponent in REG1.
0D1CH	Decrement the value of the memory pointer in register pair HL so that it points to the MSB in REG1.
0D1DH	Combine the value of the MSB of the double precision value in REG1 at the location of the memory pointer in register pair HL with the value of the sign for the result in register A.

0D1EH	Save the adjusted MSB of the double precision value in register A at the location of the memory pointer in register pair HL.
0D1FH	Return.
0D20H - 0D32H	**DOUBLE PRECISION MATH ROUTINE**
0D20H - 0D22H	Load register pair HL with the starting address in REG1.
0D23H - 0D24H	Load register B with the number of bytes to be bumped for the double precision value in REG1.
0D25H	Bump the value at the location of the memory pointer in register pair HL.
0D26H	Return if the value at the location of the memory pointer in register pair HL isn't equal to zero.
0D27H	Bump the value of the memory pointer in register pair HL.
0D28H	Decrement the value of the byte counter in register B.
0D29H - 0D2AH	Loop till all of the necessary bytes have been bumped.
0D2BH	Bump the value of the exponent at the location of the memory pointer in register pair HL.
0D2CH - 0D2EH	Go to the Level II BASIC error routine and display an OV ERROR message if the exponent for the double precision value in REG1 is too large.
0D2FH	Decrement the value of the memory pointer in register pair HL.
0D30H - 0D31H	Save a new MSB at the location of the memory pointer in register pair HL.
0D32H	Return.
0D33H - 0D44H	**DOUBLE PRECISION MATH ROUTINE**
0D33H - 0D35H	Load register pair HL with the starting address of REG2.
0D36H - 0D38H	Load register pair DE with the starting address of REG1.
0D39H - 0D3AH	Load register C with the number of bytes to be added.
0D3BH	Clear the Carry flag.

0D3CH	Load register A with the value in REG1 at the location of the memory pointer in register pair DE.
0D3DH	Add the value in REG2 at the location of the memory value in register A.
0D3EH	Save the result in register A in REG1 at the location of the memory pointer in register pair DE.
0D3FH	Bump the value of the memory pointer in register pair DE.
0D40H	Bump the value of the memory pointer in register pair HL.
0D41H	Decrement the number of bytes to be added in register C.
0D42H - 0D43H	Loop till all of the bytes for the double precision values have been added.
0D44H	Return.
0D45H - 0D56H	**DOUBLE PRECISION MATH ROUTINE**
0D45H - 0D47H	Load register pair HL with the starting address of REG2.
0D48H - 0D4AH	Load register pair DE with the starting address of REG1.
0D4BH - 0D4CH	Load register C with the number of bytes to be subtracted.
0D4DH	Clear the Carry flag.
0D4EH	Load register A with the value in REG1 at the location of the memory pointer in register pair DE.
0D4FH	Subtract the value in REG2 at the location of the memory pointer in register pair HL from the value in register A.
0D50H	Save the result in register A in REG1 at the location of the memory pointer in register pair DE.
0D51H	Bump the value of the memory pointer in register pair DE.
0D52H	Bump the value of the memory pointer in register pair HL.
0D53H	Decrement the number of bytes to be subtracted for the double precision values in register C.
0D54H - 0D55H	Loop till all of the bytes for the double precision values have been subtracted.

0D56H	Return.
0D57H - 0D68H	**DOUBLE PRECISION MATH ROUTINE**
0D57H	Load register A with the value of the sign at the location of the memory pointer in register pair HL.
0D58H	Complement the value of the sign in register A.
0D59H	Save the value of the sign in register A at the location of the memory pointer in register pair HL.
0D5AH - 0D5CH	Load register pair HL with the starting address of REG1 minus one.
0D5DH - 0D5EH	Load register B with the number of bytes to be reversed.
0D5FH	Zero register A.
0D60H	Load register C with the value in register A.
0D61H	Load register A with the value in register C.
0D62H	Subtract the value at the location of the memory pointer in register pair HL from the value in register A.
0D63H	Save the reversed value in register A at the location of the memory pointer in register pair HL.
0D64H	Bump the value of the memory pointer in register pair HL.
0D65H	Decrement the number of bytes to be reversed in register B.
0D66H - 0D67H	Loop till all of the bytes for the double precision number in REG1 have been reversed.
0D68H	Return.
0D69H - 0D8FH	**DOUBLE PRECISION MATH ROUTINE**
0D69H	Save the MSB of the double precision value in register C at the location of the memory pointer in register pair HL.
0D6AH	Save the value of the memory pointer in register pair HL on the stack.
0D6BH - 0D6CH	Subtract 8 from the number of bits to be shifted from the number of bits to be shifted in register A.
0D6DH - 0D6EH	Jump if there are less than 8 bits to be shifted.

0D6FH	Get the value of the memory pointer from the stack and put it in register pair HL.
0D70H	Save the value of the memory pointer in register pair HL on the stack.
0D71H - 0D73H	Load register pair DE with the number of bytes to be shifted and the value to be shifted into the double precision value.
0D74H	Load register C with the value at the location of the memory pointer in register pair HL.
0D75H	Save the value in register E at the location of the memory pointer in register pair HL.
0D76H	Load register E with the value in register C.
0D77H	Decrement the value of the memory pointer in register pair HL.
0D78H	Decrement the number of bytes shifted in register D.
0D79H - 0D7AH	Loop till all of the bytes have been shifted.
0D7BH - 0D7CH	Loop till there are less than 8 bits to be shifted.
0D7DH - 0D7EH	Adjust the number of bits to be shifted in register A so that it holds the correct value.
0D7FH	Load register D with the number of bits to be shifted in register A.
0D80H	Zero register A.
0D81H	Get the value of the memory pointer from the stack and put it in register pair HL.
0D82H	Decrement the number of bits to be shifted in register D.
0D83H	Return if all of the bits have been shifted.
0D84H	Save the value of the memory pointer in register pair HL on the stack.
0D85H - 0D86H	Load register E with the number of bytes to be shifted.
0D87H	Load register A with the value at the location of the memory pointer in register pair HL.
0D88H	Shift the value in register A.
0D89H	Save the adjusted value in register A at the location of the memory pointer in register pair HL.

0D8AH	Decrement the value of the memory pointer in register pair HL.
0D8BH	Decrement the number of bytes to be shifted in register E.
0D8CH - 0D8DH	Loop till all of the bytes have been shifted.
0D8EH - 0D8FH	Loop till all of the bits have been shifted.
0D90H - 0D96H	**DOUBLE PRECISION MATH ROUTINE**
0D90H - 0D92H	Load register pair HL with the address of the MSB of the double precision value in REG1.
0D93H - 0D94H	Load register D with the number of bits to be shifted.
0D95H - 0D96H	Jump.
0D97H - 0DA0H	**DOUBLE PRECISION MATH ROUTINE**
0D97H - 0D98H	Load register C with the number of bytes to be shifted.
0D99H	Load register A with the value at the location of the memory pointer in register pair HL.
0D9AH	Shift the value in register A.
0D9BH	Save the shifted value in register A at the location of the memory pointer in register pair HL.
0D9CH	Bump the value of the memory pointer in register pair HL.
0D9DH	Decrement the byte counter in register C.
0D9EH - 0D9FH	Loop till all of the bytes have been shifted.
0DA0H	Return.
0DA1H - 0DD3H	**DOUBLE PRECISION MULTIPLICATION**
0DA1H - 0DA3H	Go check to see if the value in REG1 is equal to zero.
0DA4H	Return if the double precision value in REG1 is equal to zero.
0DA5H - 0DA7H	Go figure the new exponent.
0DA8H - 0DAAH	Go move the double precision value in REG1 to a temporary work area.
0DABH	Zero the location at the current value of the memory pointer in register pair HL.

0DACH		Bump register pair DE so that it points to the LSB of the double precision value in the temporary work area.
0DADH	- 0DAEH	Load register B with the number of bytes to be figured.
0DAFH		Load register A with the value at the location of the memory pointer in register pair DE.
0DB0H		Bump the value of the memory pointer in register pair DE.
0DB1H		Check to see if the value in register A is equal to zero.
0DB2H		Save the value of the memory pointer in register pair DE on the stack.
0DB3H	- 0DB4H	Jump if the value in register A is equal to zero.
0DB5H	- 0DB6H	Load register C with the number of bits to be shifted.
0DB7H		Save the value in register pair BC on the stack.
0DB8H		Shift the value in register A one place to the right.
0DB9H		Load register B with the value in register A.
0DBAH	- 0DBCH	Go add the value in REG2 to the total in REG1 if bit zero of register A was set.
0DBDH	- 0DBFH	Go shift the value of the total in REG1.
0DC0H		Load register A with the adjusted value in register B.
0DC1H		Get the value from the stack and put it in register pair BC.
0DC2H		Decrement the number of bits to be shifted in register C.
0DC3H	- 0DC4H	Loop until all of the bits have been shifted.
0DC5H		Get the value from the stack and put it in register pair DE.
0DC6H		Decrement the number of bytes to be figured in register B.
0DC7H	- 0DC8H	Loop until all of the bytes have been figured.
0DC9H	- 0DCBH	Jump.
0DCCH	- 0DCEH	Load register pair HL with the address of the MSB in REG1.

0DCFH - 0DD1H	Go shift the double precision total in REG1.
0DD2H - 0DD3H	Jump.
0DD4H - 0DDBH	**DOUBLE PRECISION CONSTANT STORAGE AREA**
0DD4H - 0DDBH	A double precision constant equal to 10.0 is stored here.
0DDCH - 0DE4H	**DOUBLE PRECISION MATH ROUTINE**
0DDCH - 0DDEH	Load register pair DE with the starting address of the double precision constant stored above.
0DDFH - 0DE1H	Load register pair HL with the starting address of REG2.
0DE2H - 0DE4H	Go move the double precision constant into REG2.
0DE5H - 0E38H	**DOUBLE PRECISION DIVISION**
0DE5H - 0DE7H	Load register A with the value of the exponent for the double precision value in REG2.
0DE8H	Check to see if the double precision value in REG2 is equal to zero.
0DE9H - 0DEBH	Go to the Level II BASIC error routine and display a /0 ERROR message if the double precision value in REG2 is equal to zero.
0DECH - 0DEEH	Go figure the value of the new exponent.
0DEFH	Bump the value of the exponent in REG1.
0DF0H	Bump the value of the exponent in REG1.
0DF1H - 0DF3H	Go move the double precision value from REG1 into the temporary work area.
0DF4H - 0DF6H	Load register pair HL with the address of the exponent in the temporary work area.
0DF7H	Zero the value of the exponent in the temporary work area at the location of the memory pointer in register pair HL.
0DF8H	Zero register B.
0DF9H - 0DFBH	Load register pair DE with the starting address of the double precision value in the temporary work area.
0DFCH - 0DFEH	Load register pair HL with the starting address of the double precision value in REG2.
0DFFH - 0E01H	Go subtract the double precision value in REG2 from

the double precision value in the temporary work area.

0E02H	Load register A with the value at the location of the memory pointer in register pair DE.
0E03H	Subtract the value in register C from the value in register A.
0E04H	Complement the value of the Carry flag.
0E05H - 0E06H	Jump if the double precision value in REG2 is greater than the double precision value in the temporary work area.
0E07H - 0E09H	Load register pair DE with the starting address of the double precision value in the temporary work area.
0E0AH - 0E0CH	Load register pair HL with the starting address of the double precision value in REG2.
0E0DH - 0E0FH	Go add the double precision value in REG2 to the double precision value in the temporary work area.
0E10H	Zero register A.
0E12H	Save the value of the exponent in register A at the location of the memory pointer in register pair DE.
0E13H	Bump the counter in register B.
0E14H - 0E16H	Load register A with the value of the MSB of the total in REG1.
0E17H	Bump the value of the MSB for the double precision result in register A.
0E18H	Decrement the value of the MSB for the double precision result in register A.
0E19H	Put the value of the Carry flag into register A as the sign bit.
0E1AH - 0E1CH	Jump if the sign bit was set.
0E1DH	Put the sign bit in register A back into the Carry flag.
0E1EH - 0E20H	Load register pair HL with the starting address of the double precision result in REG1.
0E21H - 0E22H	Load register C with the number of bytes to be shifted.
0E23H - 0E25H	Go shift the double precision result in REG1.
0E26H - 0E28H	Load register pair HL with the starting address of

	the double precision value in the temporary work area.
0E29H - 0E2BH	Go shift the double precision value in the temporary work area.
0E2CH	Load register A with the value of the counter in register B.
0E2DH	Check to see if the value of the counter in register A is equal to zero.
0E2EH - 0E2FH	Loop till done if the value of the counter in register A isn't equal to zero.
0E30H - 0E32H	Load register pair HL with the address of the exponent for the double precision result in REG1.
0E33H	Decrement the value of the exponent for the double precision result in REG1 at the location of the memory pointer in register pair HL.
0E34H - 0E35H	Loop till done if overflow didn't occur.
0E36H - 0E38H	Go to the Level II BASIC error routine and display an OV ERROR message if the exponent for the result in REG1 is too small.
0E39H - 0E4CH	**DOUBLE PRECISION MATH ROUTINE**
0E39H	Load register A with the MSB of the double precision value in REG2 in register C.
0E3AH - 0E3CH	Save the MSB of the double precision value in REG2 in register A.
0E3DH	Decrement the value of the memory pointer in register pair HL.
0E3EH - 0E40H	Load register pair DE with the starting address of the temporary work area.
0E41H - 0E43H	Load register B with the number of bytes to be moved and zero register C.
0E44H	Load register A with the value at the location of the memory pointer in register pair HL.
0E45H	Save the value in register A at the location of the memory pointer in register pair DE.
0E46H	Zero the location of the memory pointer in register pair HL.
0E47	Decrement the value of the memory pointer in register pair DE.

0E48H	Decrement the value of the memory pointer in register pair HL.
0E49H	Decrement the value of the byte counter in register B.
0E4AH - 0E4BH	Loop till the double precision value has been moved to the temporary work area.
0E4CH	Return.
0E4DH - 0E64H	**LEVEL II BASIC MATH ROUTINE**
0E4DH - 0E4FH	Go move the value in REG1 to REG2.
0E50H	Load register pair HL with the value of the memory pointer in register pair DE.
0E51H	Decrement the value of the memory pointer in register pair HL.
0E52H	Load register A with the value of the exponent in REG1 at the location of the memory pointer in register pair HL.
0E53H	Check to see if the value in REG1 is equal to zero.
0E54H	Return if the value in REG1 is equal to zero.
0E55H - 0E56H	Add two to the value of the exponent in register A.
0E57H - 0E59H	Go to the Level II BASIC error routine and display an OV ERROR message if the adjusted exponent in register A is too large.
0E5AH	Save the adjusted exponent in register A in REG1 at the location of the memory pointer in register pair HL.
0E5BH	Save the value of the memory pointer in register pair HL on the stack.
0E5CH - 0E5EH	Go add the double precision value in REG2 to the double precision value in REG1.
0E5FH	Get the value of the memory pointer from the stack and put it in register pair HL.
0E60H	Bump the value of the exponent for the double precision value in REG1 at the location of the memory pointer in register pair HL.
0E61H	Return if overflow didn't occur.
0E62H - 0E64H	Go to the Level II BASIC error routine and display an OV ERROR message if the exponent in REG1 at

the location of the memory pointer in register pair HL is too large.

0E65H - 0F88H **ASCII TO BINARY**

0E65H - 0E67H Go zero the exponent in REG1.

0E68H - 0E6AH Go set the current number type flag to double precision.

0E6CH Zero register A.

0E6DH Load register pair DE with the current input buffer pointer in register pair HL.

0E6EH - 0E70H Load register pair BC with a zero and a negative one.

0E71H Load register H with zero.

0E72H Load register L with zero.

0E73H - 0E75H Go set the current number type flag to integer if the routine was entered from OE6CH.

0E76H Load register pair HL with the input buffer pointer in register pair DE.

0E77H Load register A with the character at the location of the current input buffer pointer in register pair HL.

0E78H - 0E79H Check to see if the character at the location of the current input buffer pointer in register A is a minus sign (–).

0E7AH Save the value in register pair AF on the stack.

0E7BH - 0E7DH Jump if the character at the location of the current input buffer pointer in register A is a minus sign (–).

0E7EH - 0E7FH Check to see if the character at the location of the current input buffer pointer in register A is a plus sign (+).

0E80H - 0E81H Jump if the character at the location of the current input buffer pointer in register A is a plus sign (+).

0E82H Decrement the value of the current input buffer pointer in register pair HL.

0E83H Go bump the current input buffer pointer in register pair HL till it points to the next character.

0E84H - 0E86H Jump if the character at the location of the current input buffer pointer in register A is numeric.

0E87H - 0E88H Check to see if the character at the location of the

current input buffer pointer in register A is a decimal point (.).

0E89H - 0E8BH	Jump if the character at the location of the current input buffer pointer in register A is a decimal point (.).
0E8CH - 0E8DH	Check to see if the character at the location of the current input buffer pointer in register A is an E.
0E8EH - 0E8FH	Jump if the character at the location of the current input buffer pointer in register A is an E.
0E90H - 0E91H	Check to see if the character at the location of the current input buffer pointer in register A is a %.
0E92H - 0E94H	Jump if the character at the location of the current input buffer pointer in register A is a %.
0E95H - 0E96H	Check to see if the character at the location of the current input buffer pointer in register A is a #.
0E97H - 0E99H	Jump if the character at the location of the current input buffer pointer in register A is a #.
0E9AH - 0E9BH	Check to see if the character at the location of the current input buffer pointer in register A is an !.
0E9CH - 0E9EH	Jump if the character at the location of the current input buffer pointer in register A is an !.
0E9FH - 0EA0H	Check to see if the character at the location of the current input buffer pointer in register A is a D.
0EA1H - 0EA2H	Jump if the character at the location of the current input buffer pointer in register A isn't a D.
0EA3H	Set the flags according to the value of the character at the location of the current input buffer pointer in register A.
0EA4H - 0EA6H	Go convert the current value in REG1 to either single precision or double precision.
0EA7H	Save the current input buffer pointer in register pair HL on the stack.
0EA8H - 0EAAH	Load register pair HL with the return address.
0EABH	Exchange the return address in register pair HL with the value of the current input buffer pointer in register pair HL.
0EACH	Go bump the current input buffer pointer in register pair HL till it points to the next character.
0EADH	Decrement the value in register D.

0EAEH - 0EAFH	Check to see if the character at the location of the current input buffer pointer in register A is a minus sign token (CEH).
0EB0H	Return if the character at the location of the current input buffer pointer in register A is a minus sign token (CEH).
0EB1H - 0EB2H	Check to see if the character at the location of the current input buffer pointer in register A is a minus sign (–).
0EB3H	Return if the character at the location of the current input buffer pointer in register A is a minus sign (–).
0EB4H	Bump the value in register D.
0EB5H - 0EB6H	Check to see if the character at the location of the current input buffer pointer in register A is a plus sign token (CDH.)
0EB7H	Return if the character at the location of the current input buffer pointer in register A is a plus sign token (CDH).
0EB8H - 0EB9H	Check to see if the character at the location of the current input buffer pointer in register A is a plus sign (+).
0EBBH	Return if the character at the location of the current input buffer pointer in register A is a plus sign (+).
0EBAH	Decrement the value of the input buffer pointer in register pair HL.
0EBCH	Get the value from the stack and put it in register pair AF.
0EBDH	Go bump the current input buffer pointer in register pair HL till it points to the next character.
0EBEH - 0EC0H	Jump if the character at the location of the input buffer pointer in register A is numeric.
0EC1H	Bump the value in register D.
0EC2H - 0EC3H	Jump if the exponent is positive.
0EC4H	Zero register A.
0EC5H	Subtract the value of the exponent in register E from register A.
0EC6H	Load register E with the value of the exponent in register A.

0EC7H	Save the current input buffer pointer in register pair HL on the stack.
0EC8H	Load register A with the value of the exponent in register E.
0EC9H	Subtract the value in register B from the exponent in register A.
0ECAH - 0ECCH	Go multiply the current value by ten if necessary.
0ECDH - 0ECFH	Go divide the current value by ten if necessary.
0ED0H - 0ED1H	Loop till the value is adjusted correctly.
0ED2H	Get the value of the current input buffer pointer from the stack and put it in register pair HL.
0ED3H	Get the value from the stack and put it in register pair AF.
0ED4H	Save the value of the current input buffer pointer in register pair HL on the stack.
0ED5H - 0ED7H	Go convert the current value to negative if necessary.
0ED8H	Get the value of the current input buffer pointer from the stack and put it in register pair HL.
0ED9H	Go check the current value of the number type flag.
0EDAH	Return if the current value of the number type flag isn't single precision.
0EDBH	Save the current value of the input buffer pointer in register pair HL on the stack.
0EDCH - 0EDEH	Load register pair HL with the return address.
0EDFH	Save the value of the return address in register pair HL on the stack.
0EE0H - 0EE2H	Go convert the current value in REG1 to an integer if possible.
0EE3H	Return.
0EE4H	Go check the current value of the number type flag.
0EE5H	Bump the value in register C.
0EE6H - 0EE7H	Jump if no need to convert the current value in REG1.
0EE8H - 0EEAH	Go convert the current value in REG1 to single

	precision if the current value isn't double precision.
0EEBH - 0EEDH	Jump.
0EEEH	Go check the current value of the number type flag.
0EEFH - 0EF1H	Go to the Level II BASIC error routine and display a SN ERROR message if the current value in REG1 is a single precision value or a double precision value.
0EF2H	Bump the value of the current input buffer pointer in register pair HL.
0EF3H - 0EF4H	Jump.
0EF5H	Set the flags according to the character at the location of the current input buffer pointer in register pair HL.
0EF6H - 0EF8H	Go convert the current value in REG1 to either single precision or double precision.
0EF9H - 0EFAH	Jump.
0EFBH	Save the value in register pair HL on the stack.
0EFCH	Save the value in register pair DE on the stack.
0EFDH	Save the value in register pair BC on the stack.
0EFEH	Save the value in register pair AF on the stack.
0EFFH - 0F01H	Go convert the current value in REG1 to single precision if necessary.
0F02H	Get the value from the stack and put it in register pair AF.
0F03H - 0F05H	Go convert the current value in REG1 to double precision if necessary.
0F06H	Get the value from the stack and put it in register pair BC.
0F07H	Get the value from the stack and put it in register pair DE.
0F08H	Get the value from the stack and put it in register pair HL.
0F09H	Return.
0F0AH	Return if the current value is an integer.
0F0BH	Save the value in register pair AF on the stack.
0F0CH	Go check the current value of the number type flag.

0F0DH	Save the value in register pair AF on the stack.
0F0EH - 0F10H	Go multiply the current value in REG1 by ten if it's single precision.
0F11H	Get the value from the stack and put it in register pair AF.
0F12H - 0F14H	Go multiply the current value in REG1 by ten if it's double precision.
0F15H	Get the value from the stack and put it in register pair AF.
0F16H	Decrement the value in register A.
0F17H	Return.
0F18H	Save the value in register pair DE on the stack.
0F19H	Save the value in register pair HL on the stack.
0F1AH	Save the value in register pair AF on the stack.
0F1BH	Go check the current value of the number type flag.
0F1CH	Save the value in register pair AF on the stack.
0F1DH - 0F1FH	Go divide the current value in REG1 by 10 if the current number type is single precision.
0F20H	Get the value from the stack and put it in register pair AF.
0F21H - 0F23H	Go divide the current value in REG1 by 10 if the current number type is double precision.
0F24H	Get the value from the stack and put it in register pair AF.
0F25H	Get the value from the stack and put it in register pair HL.
0F26H	Get the value from the stack and put it in register pair DE.
0F27H	Bump the value in register A.
0F28H	Return.
0F29H	Save the value in register pair DE on the stack.
0F2AH	Load register A with the value in register B.
0F2BH	Add the value in register C to the value in register A.

0F2CH	Load register B with the adjusted value in register A.
0F2DH	Save the value in register pair BC on the stack.
0F2EH	Save the value in register pair HL on the stack.
0F2FH	Load register A with the character at the location of the current input buffer pointer in register pair HL.
0F30H - 0F31H	Subtract 30H from the ASCII value in register A so that it will be binary.
0F32H	Save the adjusted value in register A on the stack.
0F33H	Go check the current value of the number type flag.
0F34H - 0F36H	Jump if the current value in REG1 isn't an integer.
0F37H - 0F39H	Load register pair HL with the integer value in REG1.
0F3AH - 0F3CH	Load register pair DE with 3277.
0F3DH	Go check to see if the integer value in register pair HL is greater than or equal to 3277.
0F3EH - 0F3FH	Jump if the integer value in register pair HL is greater than or equal to 3277.
0F40H	Load register D with the MSB of the integer value in register H.
0F41H	Load register E with the LSB of the integer value in register L.
0F42H	Multiply the integer value in register pair HL by two.
0F43H	Multiply the integer value in register pair HL by two. Register pair HL now holds the original integer value times four.
0F44H	Add the original integer value in register pair DE to the integer value in register pair HL. Register pair HL now holds the original integer value times five.
0F45H	Multiply the integer value in register pair HL by two. Register pair HL now holds the original integer value times ten.
0F46H	Get the binary value for the next character from the stack and put it in register A.
0F47H	Load register C with the value of the character in register A.

0F48H	Add the value of the character in register pair BC to the integer value in register pair HL.
0F49H	Load register A with the MSB of the integer value in register H.
0F4AH	Check to see if the sign bit is set.
0F4BH - 0F4DH	Jump if the sign bit is set in the integer value in register pair HL.
0F4EH - 0F50H	Save the integer value in register pair HL as the current integer value in REG1.
0F51H	Get the value from the stack and put it in register pair HL.
0F52H	Get the value from the stack and put it in register pair BC.
0F53H	Get the value from the stack and put it in register pair DE.
0F54H - 0F56H	Jump.
0F57H	Load register A with the binary value of the character in register C.
0F58H	Save the value in register A on the stack.
0F59H - 0F5BH	Go convert the current value in REG1 to single precision.
0F5CH	Set the Carry flag.
0F5DH - 0F5EH	Jump if the current value in REG1 is double precision.
0F5FH - 0F61H	Load register pair BC with the exponent and the MSB of a single precision constant.
0F62H - 0F64H	Load register pair DE with the NMSB and the LSB of a single precision constant. Register pairs BC and DE now hold a single precision constant equal to 1E6.
0F65H - 0F67H	Check to see if the single precision value in REG1 is greater than or equal to 1E6.
0F68H - 0F6AH	Jump if the single precision value in REG1 is greater than or equal to 1E6.
0F6BH - 0F6DH	Go multiply the single precision value in REG1 by 10.
0F6EH	Get the binary value of the character from the stack and put it in register A.

0F6FH - 0F71H	Go add the value in register A to the single precision value in REG1.
0F72H - 0F73H	Jump.
0F74H - 0F76H	Go convert the single precision value in REG1 to double precision.
0F77H - 0F79H	Go multiply the double precision value in REG1 by ten.
0F7AH - 0F7CH	Go move the double precision value in REG1 to REG2.
0F7DH	Get the binary value for the character from the stack and put it in register A.
0F7EH - 0F80H	Go convert the binary value in register A to single precision.
0F81H - 0F83H	Go convert the single precision value in REG1 to double precision.
0F84H - 0F86H	Go add the double precision value in REG2 to the double precision value in REG1.
0F87H - 0F88H	Jump.
0F89H - 0F93H	**SINGLE PRECISION MATH ROUTINE**
0F89H - 0F8BH	Go move the current single precision value in REG1 to the stack.
0F8CH - 0F8EH	Go convert the value in register A to single precision and return with the result in REG1.
0F8FH	Get the exponent and the MSB for the single precision value from the stack and put it in register pair BC.
0F90H	Get the NMSB and the LSB of the single precision value from the stack and put it in register pair DE.
0F91H - 0F93H	Go add the single precision value in REG1 to the single precision value in register pairs BC and DE.
0F94H - 0FA6H	**LEVEL II BASIC MATH ROUTINE**
0F94H	Load register A with the value of the exponent in register E.
0F95H - 0F96H	Check to see if the value of the exponent in register A is greater than or equal to 10.
0F97H - 0F98H	Jump if the value of the exponent in register A is greater than or equal to 10.

0F99H		Multiply the value in register A by two.
0F9AH		Multiply the value in register A by two. Register A now holds the original value of the exponent times four.
0F9BH		Add the original value of the exponent in register E to the adjusted value of the exponent in register A.
0F9CH		Multiply the value in register A by two. Register A now holds the original value of the exponent times ten.
0F9DH		Add the value of the current character at the location of the input buffer pointer in register pair HL to the adjusted value in register A.
0F9EH -	0F9FH	Convert the adjusted value in register A to it's binary equivalent.
0FA0H		Load register E with the adjusted value in register A.
0FA2H -	0FA3H	Load register E with the maximum exponent.
0FA4H -	0FA6H	Jump.
0FA7H -	0FAEH	**DISPLAY MESSAGE ROUTINE**
0FA7H		Save the value in register pair HL on the stack.
0FA8H -	0FAAH	Load register pair HL with the starting address of the IN message.
0FABH -	0FADH	Go display the message pointed to by register pair HL.
0FAEH		Get the value from the stack and put it in register pair HL.
0FAFH -	0FBCH	**CONVERT BINARY TO ASCII AND DISPLAY RESULT**
0FAFH -	0FB1H	Go save the value in register pair HL as the current value in REG1.
0FB2H		Zero register A.
0FB3H -	0FB5H	Go initialize the input buffer for the ASCII conversion.
0FB6H		Set the flags.
0FB7H -	0FB9H	Go convert the integer value in REG1 to an ASCII string. Return with register pair HL pointing to the result.

0FBAH - 0FBCH	Go display the message pointed to by register pair HL.
0FBDH - 1363H	**BINARY TO ASCII CONVERSION ROUTINE**
0FBDH	Zero register A.
0FBEH - 0FC0H	Go initialize the input buffer for the ASCII conversion.
0FC1H - 0FC2H	Check the value of the edit flag in register A to see if the sign is to be included.
0FC3H - 0FC4H	Jump if the sign isn't to be included in the ASCII string.
0FC5H - 0FC6H	Save a plus sign (+) at the location of the input buffer pointer in register pair HL.
0FC7H	Load register pair DE with the value of the input buffer pointer in register pair HL.
0FC8H - 0FCAH	Go determine the value of the sign for the current value in REG1.
0FCBH	Load register pair HL with the input buffer pointer in register pair DE.
0FCCH - 0FCEH	Jump if the current value in REG1 is positive.
0FCFH - 0FD0H	Save a minus sign (–) at the location of the input buffer pointer in register pair HL.
0FD1H	Save the value in register pair BC on the stack.
0FD2H	Save the value in register pair HL on the stack.
0FD3H - 0FD5H	Go convert the negative value in REG1 to it's positive equivalent.
0FD6H	Get the value from the stack and put it in register pair HL.
0FD7H	Get the value from the stack and put it in register pair HL.
0FD8H	Set the flags.
0FD9H	Bump the input buffer pointer in register pair HL.
0FDAH - 0FDBH	Save an ASCII zero (0) at the location of the input buffer pointer in register pair HL.
0FDCH - 0FDEH	Load register A with the value of the edit flag.
0FDFH	Load register D with the value of the edit flag in register A.

0FE0H	Move the edit on/off bit in register A into the Carry flag.
0FE1H - 0FE3H	Load register A with the value of the current number type flag.
0FE4H - 0FE6H	Jump if value is to be edited.
0FE7H - 0FE9H	Jump if no edit.
0FEAH - 0FEBH	Check to see if the current value in REG1 is single or double precision.
0FECH - 0FEEH	Jump if the current value in REG1 is single or double precision.
0FEFH - 0FF1H	Load register pair BC with zero.
0FF2H - 0FF4H	Go convert the integer value in REG1 to an ASCII string.
0FF5H - 0FF7H	Load register pair HL with the starting address of the input buffer.
0FF8H	Load register B with the character at the location of the input buffer pointer in register pair HL.
0FF9H - 0FFAH	Load register C with a space character.
0FFBH - 0FFDH	Load register A with the value of the edit flag.
0FFEH	Load register E with the value of the edit flag in register A.
0FFFH - 1000H	Check the value of the edit flag in register A to see if leading * are to be printed.
1001H - 1002H	Jump if leading * aren't to be included in the ASCII string.
1003H	Load register A with the character at the location of the input buffer pointer in register B.
1004H	Check to see if the character at the location of the input buffer pointer in register A is a space.
1005H - 1006H	Load register C with an * character.
1007H - 1008H	Jump if the character at the location of the input buffer pointer in register A isn't a space.
1009H	Load register B with the * character in register C.
100AH	Save the * character in register C at the location of the input buffer pointer in register pair HL.

100BH	Go bump the input buffer pointer in register pair HL till it points to the next character.
100CH - 100DH	Jump if the character at the location of the input buffer pointer in register pair HL is the end of the input buffer character (00H).
100EH - 100FH	Check to see if the character at the location of the input buffer pointer in register A is an E.
1010H - 1011H	Jump if the character at the location of the input buffer pointer in register A is an E.
1012H - 1013H	Check to see if the character at the location of the input buffer pointer in register A is a D.
1014H - 1015H	Jump if the character at the location of the input buffer pointer in register A is a D.
1016H - 1017H	Check to see if the character at the location of the input buffer pointer in register A is a D.
1018H - 1019H	Jump if the character at the location of the input buffer pointer in register A is a 0.
101AH - 101BH	Check to see if the character at the location of the input buffer pointer in register A is a comma.
101CH - 101DH	Jump if the character at the location of the input buffer pointer in register A is a comma.
101EH - 101FH	Check to see if the character at the location of the input buffer pointer in register A is a decimal point.
1020H - 1021H	Jump if the character at the location of the input buffer pointer in register A isn't a decimal point.
1022H	Decrement the value of the input buffer pointer in register pair HL.
1023H - 1024H	Save a zero character (0) at the location of the input buffer pointer in register pair HL.
1025H	Load register A with the value of the edit flag in register E.
1026H - 1027H	Check to see if a $ is to be included in the ASCII string.
1028H - 1029H	Jump if a $ isn't to be included in the ASCII string.
102AH	Decrement the value of the input buffer pointer in register pair HL.
102BH - 102CH	Save a $ character at the location of the input buffer pointer in register pair HL.

102DH	Load register A with the value of the input flag in register E.
102EH - 102FH	Check to see if the sign is to follow the ASCII string.
1030H	Return if the sign isn't to follow the ASCII string.
1031H	Decrement the value of the input buffer pointer in register pair HL.
1032H	Save the character in register B at the location of the input buffer pointer in register pair HL.
1033H	Return.
1034H - 1036H	Save the value of the edit flag in register A.
1037H - 1039H	Load register pair HL with the starting address of the input buffer.
103AH - 103BH	Save a space at the location of the input buffer pointer in register pair HL.
103CH	Return.
103DH - 103EH	Set the Carry flag according to whether the current value in REG1 is single precision or double precision.
103FH	Save the value in register pair HL on the stack.
1040H - 1041H	Adjust the value of the number type in register A.
1042H	Multiply the value of the number type in register A by two.
1043H	Load register D with the adjusted value of the number type in register A.
1044H	Bump the value of the number type in register D.
1045H - 1047H	Go adjust the current value in REG1.
1048H - 104AH	Load register pair BC with a default value.
104BH	Add the value in register D to the value in register A.
104CH - 104EH	Jump if the number of digits in register A is too large.
104FH	Bump the value in register D.
1050H	Check to see if the number of digits in register A is equal to the value in register D.
1051H - 1052H	Jump if the number of digits in register A is greater than or equal to the value in register D.

1053H	Bump the number of digits in register A.
1054H	Load register B with the number of digits in register A.
1055H - 1056H	Load register A with a two.
1057H - 1058H	Make register A equal to zero.
1059H	Get the value from the stack and put it in register pair HL.
105AH	Save the value in register pair AF on the stack.
105BH - 105DH	Go put a comma or decimal point in the input buffer if necessary.
105EH - 105FH	Save a zero (0) character at the location of the input buffer pointer in register pair HL.
1060H - 1062H	Go bump the value of the input buffer pointer in register pair HL if necessary.
1063H - 1065H	Go convert the binary value in REG1 to ASCII.
1066H	Decrement the value of the input buffer pointer in register pair HL.
1067H	Load register A with the value at the location of the input buffer pointer in register pair HL.
1068H - 1069H	Check to see if the value in register A is a zero character.
106AH - 106BH	Loop till the character at the location of the input buffer pointer in register pair HL is not a zero character.
106CH - 106DH	Check to see if the character at the location of the input buffer pointer in register A is a decimal point.
106EH - 1070H	Go bump the value of the input buffer pointer in register pair HL if the character in register A isn't a decimal point.
1071H	Get the value from the stack and put it in register pair HL.
1072H - 10731	Jump if done.
1074H	Save the value in register pair AF on the stack.
1075H	Go check the current value of the number type flag.
1076H - 1077H	Load register A with the starting value for a D or E character.

1078H	Multiply the value of the character in register A by two and add in the value of the Carry flag from the number type flag test. Register A will now be equal to a D or an E depending on the current value of the number type flag.
1079H	Save the character in register A at the location of the input buffer pointer in register pair HL.
107AH	Bump the value of the input buffer pointer in register pair HL.
107BH	Get the value of the exponent from the stack and put it in register A.
107CH - 107DH	Save a plus sign at the location of the input buffer pointer in register pair HL.
107EH - 1080H	Jump if the exponent is positive.
1081H - 1082H	Save a minus sign at the location of the input buffer pointer in register pair HL.
1083H	Reverse the value of the exponent in register A.
1084H	Bump the value of the exponent in register A so that it will be positive.
1085H - 1086H	Load register B with an ASCII zero minus one.
1087H	Bump the value of the ASCII character in register B.
1088H - 1089H	Subtract ten from the value of the exponent in register A.
108AH - 108BH	Loop till the value of the exponent in register A is less than ten.
108CH - 108DH	Adjust the value of the remainder in register A so that it will be an ASCII character.
108EH	Bump the value of the input buffer pointer in register pair HL.
108FH	Save the ASCII character in register B at the location of the input buffer pointer in register pair HL.
1090H	Bump the value of the input buffer pointer in register pair HL.
1091H	Save the value of the ASCII character in register A at the location of the input buffer pointer in register pair HL.
1092H	Bump the value of the input buffer pointer in register pair HL.

1093H - 1094H	Save an end of the ASCII string character at the location of the input buffer pointer in register pair HL.
1095H	Load register pair DE with the value of the input buffer pointer in register pair HL.
1096H - 1098H	Load register pair HL with the starting address of the ASCII string.
1099H	Return.
109AH	Bump the value of the input buffer pointer in register pair HL.
109BH	Save the value in register pair BC on the stack.
109CH - 109DH	Check to see if the current number type in REG1 is single or double precision.
109EH	Load register A with the value of the edit flag in register D.
109FH - 10A1H	Jump if the current value in REG1 is either single precision or double precision.
10A2H	Put the value of the exponential notation flag into the Carry flag.
10A3H - 10A5H	Jump if exponential notation is requested.
10A6H - 10A8H	Load register pair BC with a default value.
10A9H - 10ABH	Go check to see if commas are requested.
10ACH	Get the value from the stack and put it in register pair DE.
10ADH	Load register A with the number of digits requested to the left of the decimal point in register D.
10AEH - 10AFH	Subtract the maximum number of digits to the left of the decimal point from the number of digits to the left of the decimal point requested.
10B0H - 10B2H	Go put leading zeros into the input buffer if the number of digits to the left of the decimal point requested is greater than the number of digits to the left of the decimal point for the maximum length of an integer value.
10B3H - 10B5H	Go convert the integer value in REG1 to an ASCII string.
10B6H	Load register A with the number of digits to the right of the decimal point requested in register E.

10B7H	Check to see if there are any digits to the right of the decimal point requested.
10B8H - 10BAH	Go decrement the value of the input buffer pointer in register pair HL if there are no digits to the right of the decimal point requested.
10BBH	Decrement the number of digits to the right of the decimal point in register A.
10BCH - 10BEH	Go add trailing zeros to the ASCII string if necessary.
10BFH	Save the value in register pair HL on the stack.
10C0H - 10C2H	Go edit the ASCII string in the input buffer.
10C3H	Get the value from the stack and put it in register pair HL.
10C4H - 10C5H	Jump if the sign follows the ASCII string.
10C6H	Save the character in register B at the location of the input buffer pointer in register pair HL.
10C7H	Bump the value of the input buffer pointer in register pair HL.
10C8H - 10C9H	Save an end of the ASCII string character at the location of the input buffer pointer in register pair HL.
10CAH - 10CCH	Load register pair HL with the starting address of the input buffer pointer.
10CDH	Bump the value of the input buffer pointer in register pair HL.
10CEH - 10D0H	Load register A with the LSB of the address of the decimal point for the ASCII string.
10D1H	Subtract the LSB of the input buffer pointer address in register L from the value in register A.
10D2H	Subtract the number of digits to the left of the decimal point in register D from the adjusted value in register A.
10D3H	Return if the input buffer pointer in register pair HL points to the start of the ASCII string.
10D4H	Load register A with the character at the location of the input buffer pointer in register pair HL.
10D5H - 10D6H	Check to see if the character at the location of the input buffer pointer in register A is a space.

10D7H - 10D8H	Jump if the character at the location of the input buffer pointer in register pair HL is a space.
10D9H - 10DAH	Check to see if the character at the location of the input buffer pointer in register A is a *.
10DBH - 10DCH	Jump if the character at the location of the input buffer pointer in register A is a *.
10DDH	Decrement the value of the input buffer pointer in register pair HL.
10DEH	Save the value of the input buffer pointer in register pair HL on the stack.
10DFH	Save the value in register pair AF on the stack.
10E0H - 10E2H	Load register pair BC with the return address.
10E3H	Save the return address in register pair BC on the stack.
10E4H	Go bump the value of the input buffer pointer in register pair HL till it points to the next character.
10E5H - 10E6H	Check to see if the character at the location of the input buffer pointer in register A is a minus sign.
10E7H	Return if the character at the location of the input buffer pointer in register A is a minus sign.
10E8H - 10E9H	Check to see if the character at the location of the input buffer pointer in register A is a plus sign.
10EAH	Return if the character at the location of the input buffer pointer in register A is a plus sign.
10EBH - 10ECH	Check to see if the character at the location of the input buffer pointer in register A is a dollar sign.
10EDH	Return if the character at the location of the input buffer pointer in register A is a dollar sign.
10EEH	Get the return address from the stack and put it in register pair BC.
10EFH - 10F0H	Check to see if the character at the location of the input buffer pointer in register A is an ASCII zero.
10F1H - 10F2H	Jump if the character at the location of the input buffer pointer in register A isn't an ASCII zero.
10F3H	Bump the value of the input buffer pointer in register pair HL.
10F4H	Go bump the value of the input buffer pointer in register pair HL till it points to the next character.

10F5H - 10F6H	Jump if the character at the location of the input buffer pointer in register A isn't a numeric character.
10F7H	Decrement the value of the input buffer pointer in register pair HL.
10F9H	Decrement the value of the input buffer pointer in register pair HL.
10FAH	Save the character in register A at the location of the input buffer pointer in register pair HL.
10FBH	Get the value from the stack and put it in register pair AF.
10FCH - 10FDH	Loop till done.
10FEH	Get the value from the stack and put it in register pair BC.
10FFH - 1101H	Jump.
1102H	Get the value from the stack and put it in register pair AF.
1103H - 1104H	Loop till done.
1105H	Get the value from the stack and put it in register pair HL.
1106H - 1107H	Save a % character at the location of the input buffer pointer in register pair HL.
1108H	Return.
1109H	Save the value in register pair HL on the stack.
110AH	Check to see if exponential notation is requested by the edit flag in register A.
110BH - 110DH	Jump if exponential notation is requested by the edit flag in register A.
110EH - 110FH	Jump if the current value in REG1 is single precision.
1110H - 1112H	Load register pair DE with the address of the double precision value to be compared to the current value in REG1. Register pair DE points to a double precision constant equal to 1E16.
1113H - 1115H	Go compare the double precision constant pointed to by register pair DE to the double precision value in REG1.
1116H - 1117H	Load register D with the maximum length of a double precision value.

1118H - 111AH	Jump if the double precision value in REG1 is less than or equal to 1E16H.
111BH	Get the value from the stack and put it in register pair HL.
111CH	Get the value from the stack and put it in register pair BC.
111DH - 111FH	Go convert the double precision value in REG1 to an ASCII string.
1120H	Decrement the input buffer pointer in register pair HL.
1121H - 1122H	Save a % character at the location of the input buffer pointer in register pair HL.
1123H	Return.
1124H - 1126H	Load register pair BC with the exponent and the MSB of a single precision constant.
1127H - 1129H	Load register pair DE with the NMSB and the LSB of a single precision constant. Register pairs BC and DE now hold a single precision constant equal to 1E16.
112AH - 112CH	Go compare the single precision constant in register pairs BC and DE to the single precision value in REG1.
112DH - 112FH	Jump if the single precision value in REG1 is greater than 1E16.
1130H - 1131H	Load register D with the maximum length of a single precision value.
1132H - 1134H	Go check the sign of the value in REG1.
1135H - 1137H	Go adjust the current value of REG1.
1138H	Get the value from the stack and put it in register pair HL.
1139H	Get the value from the stack and put it in register pair BC.
113AH - 113CH	Jump if the value in REG1 had to be multiplied to get it in range.
113DH	Save the value in register pair BC on the stack.
113EH	Load register E with the number of times the current value in REG1 was divided.

113FH	Load register A with the number of digits to the left of the decimal point requested.
1140H	Subtract the maximum length for the current number type in register D from the number of digits requested in register A.
1141H	Subtract the number of times the current value in REG1 was divided in register E from the adjusted value in register A.
1142H - 1144H	Go put leading zeros into the input buffer if necessary.
1145H - 1147H	Go figure the comma count and the position of the decimal point.
1148H - 114AH	Go convert the integer portion of the current value in REG1 to an ASCII string.
114BH	Check to see if there are any trailing zeros needed.
114CH - 114EH	Go put trailing zeros into the input buffer if necessary.
114FH	Check to see if commas or the decimal point is needed.
1150H - 1152H	Go put commas and the decimal point into the input buffer if necessary.
1153H	Get the value from the stack and put it in register pair DE.
1154H - 1156H	Jump.
1157H	Load register E with the number of times the current value in REG1 was multiplied.
1158H	Load register A with the number of digits requested to the right of the decimal point.
1159H	Check to see if any digits to the right of the decimal point was requested.
115AH - 115CH	Go decrement the number of digits requested to the right of the decimal point if necessary.
115DH	Add the number of times the current value was multiplied in register E to the number of digits to the right of the decimal point requested in register A.
115EH - 1160H	Jump if the value in REG1 must be adjusted.
1161H	Zero register A.

1162H	Save the value in register pair BC on the stack.
1163H	Save the value in register pair AF on the stack.
1164H - 1166H	Go divide the value in REG1 by ten if necessary.
1167H - 1169H	Loop till the value in REG1 is properly adjusted.
116AH	Get the value from the stack and put it in register pair BC.
116BH	Load register A with the number of times the value in REG1 was multiplied in register E.
116CH	Add the value in register B to the value in register A.
116DH	Get the value from the stack and put it in register pair BC.
116EH	Load register E with the adjusted value in register A.
116FH	Add the length of the maximum size for the current value in register D to the adjusted value in register A.
1170H	Add the number of digits requested for the left of the decimal point in register B to the adjusted value in register A.
1171H - 1173H	Jump if there are no digits to the left of the decimal point.
1174H	Subtract the maximum length for the current value in register D from the adjusted value in register A.
1175H	Subtract the value in register E from the adjusted value in register A.
1176H - 1178H	Go put leading zeros into the input buffer if necessary.
1179H	Save the value in register pair BC on the stack.
117AH - 117CH	Go determine the position of the decimal point and the comma count.
117DH - 117EH	Jump.
117FH - 1181H	Go put leading zeros into the input buffer if necessary.
1182H	Load register A with the number of bytes requested to the right of the decimal point in register C.
1183H - 1185H	Go put a decimal point into the input buffer.

1186H	Load register C with the number of digits requested to the right of the decimal point in register A.
1187H	Zero register A.
1188H	Subtract the maximum length for the current value in register D from the value in register A.
1189H	Subtract the value in register E from the adjusted value in register A.
118AH - 118CH	Go put zeros into the input buffer if necessary.
118DH	Save the value in register pair BC on the stack.
118EH	Load register B with the value in register A.
118FH	Load register C with the value in register A.
1190H - 1192H	Go convert the current value in REG1 to an ASCII string.
1193H	Get the value from the stack and put it in register pair BC.
1194H	Check to see if there are any digits to the right of the decimal point requested.
1195H - 1196H	Jump if there are digits to the right of the decimal point requested.
1197H - 1199H	Load register pair HL with the position of the decimal point.
119AH	Add the value in register E to the value in register A.
119BH	Decrement the adjusted value in register A.
119CH - 119EH	Go put zeros into the input buffer if necessary.
119FH	Load register D with the number of digits to the left of the decimal point requested in register B.
11A0H - 11A2H	Jump.
11A3H	Save the value in register pair HL on the stack.
11A4H	Save the value in register pair BC on the stack.
11A5H - 11A7H	Go convert the integer value in REG1 to a single precision value.
11A8H	Get the value from the stack and put it in register pair DE.
11A9H	Zero register A.

11AAH - 11ACH	Jump if the current value in REG1 is single precision.
11ADH - 11AEH	Load register E with the maximum length of a double precision value.
11B0H - 11B1H	Load register E with the maximum length of a single precision value.
11B2H - 11B4H	Go check the sign for the current value in REG1.
11B5H	Set the Carry flag.
11B6H - 11B8H	Go adjust the current value in REG1 if it's nonzero.
11B9H	Get the value from the stack and put it in register pair HL.
11BAH	Get the value from the stack and put it in register pair BC.
11BBH	Save the value in register pair AF on the stack.
11BCH	Load register A with the number of digits to the right of the decimal point requested in register C.
11BDH	Check to see if there are any digits to the right of the decimal point requested.
11BEH	Save the value in register pair AF on the stack.
11BFH - 11C1H	Go decrement the number of digits requested to the right of the decimal point in register A if necessary.
11C2H	Add the number of digits requested for the left of the decimal point in register B to the number of digits requested to the right of the decimal point in register A.
11C3H	Load register C with the adjusted value in register A.
11C4H	Load register A with the value of the edit flag in register D.
11C5H - 11C6H	Check to see if the sign is to follow the ASCII string.
11C7H - 11C8H	Set the Carry flag according to the sign following the ASCII string test.
11C9H	Adjust register A so that it will reflect the sign following the ASCII string test.
11CAH	Load register D with the value in register A.
11CBH	Add the value in register C to the value in register A.

11CCH	Load register C with the adjusted value in register A.
11CDH	Subtract the value in register E from the adjusted value in register A.
11CEH	Save the value in register pair AF on the stack.
11CFH	Save the value in register pair BC on the stack.
11D0H - 11D2H	Go divide the current value in REG1 by ten if necessary.
11D3H - 11D5H	Loop till the current value in REG1 is properly adjusted.
11D6H	Get the value from the stack and put it in register pair BC.
11D7H	Get the value from the stack and put it in register pair AF.
11D8H	Save the value in register pair BC on the stack.
11D9H	Save the value in register pair AF on the stack.
11DAH - 11DCH	Jump if the value in register A is negative.
11DDH	Zero register A.
11DEH	Invert the value in register A.
11DFH	Bump the value in register A so that it will be positive.
11E0H	Add the number of digits requested to the left of the decimal point in register B to the adjusted value in register A.
11E1H	Bump the adjusted value in register A.
11E3H	Add the value of the maximum length for the current number type in register D to the adjusted value in register A.
11E4H - 11E5H	Load register C with zero.
11E6H - 11E8H	Go convert the current value in REG1 to an ASCII string.
11E9H	Get the value from the stack and put it in register pair AF.
11EAH - 11ECH	Go put zeros into the input buffer if necessary.
11EDH	Get the value from the stack and put it in register pair BC.

11EEH	Get the value from the stack and put it in register pair AF.
11EFH - 11F1H	Go decrement the input buffer pointer in register pair HL if necessary.
11F2H	Get the value from the stack and put it in register pair AF.
11F3H - 11F4H	Jump if the Carry flag was set during the first pass.
11F5H	Add the value in register E to the value in register A.
11F6H	Subtract the number of digits to the left of the decimal point requested in register B from the adjusted value in register A.
11F7H	Subtract the value in register D from the value in register A.
11F8H	Save the value in register pair BC on the stack.
11F9H - 11FBH	Go figure the value of the exponent for the current value in REG1.
11FCH	Load register pair HL with the value of the input buffer pointer in register pair DE.
11FDH	Get the value from the stack and put it in register pair DE.
11FEH - 1200H	Jump.
1201H	Save the value in register pair DE on the stack.
1202H	Zero register A.
1203H	Save the value in register pair AF on the stack.
1204H	Go check the current value of the number type flag.
1205H - 1207H	Jump if the current value in REG1 is single precision.
1208H - 120AH	Load register A with the value of the exponent for the double precision value in REG1.
120BH - 120CH	Check to see if the double precision value in REG1 uses more than 16 bits of precision for the integer portion of the double precision value.
120DH - 120FH	Jump if the double precision value in REG1 uses more than 16 bits of precision for it's integer portion.
1210H - 1212H	Load register pair DE with the starting address for the double precision constant equal to 1E10.

1213H - 1215H	Load register pair HL with the starting address for REG2.
1216H - 1218H	Go move the double precision constant into REG2.
1219H - 121BH	Go multiply the double precision value in REG1 by the double precision constant in REG2.
121CH	Get the value from the stack and put it in register pair AF.
121DH - 121EH	Subtract ten from the value in register A.
121FH	Save the value in register pair AF on the stack.
1220H - 1221H	Jump.
1222H - 1224H	Go compare the current value in REG1 to 999999.5.
1225H	Go check the current value of the number type flag.
1226H - 1228H	Jump if the current value in REG1 is double precision.
1229H - 122BH	Load register pair BC with the exponent and the MSB of a single precision constant.
122CH - 122EH	Load register pair DE with the NMSB and the LSB of a single precision constant. Register pairs BC and DE are now equal to a single precision constant of 99,999.945.
122FH - 1231H	Go compare the single precision value in REG1 to the single precision constant in register pairs BC and DE.
1232H - 1233H	Jump.
1234H - 1236H	Load register pair DE with the starting address of a double precision constant equal to 9.9921E14.
1237H - 1239H	Go compare the double precision constant pointed to by register pair DE to the double precision value in REG1.
123AH - 123CH	Jump if the current value in REG1 is greater than the constant compared to it.
123DH	Get the value from the stack and put it in register pair AF.
123EH - 1240H	Go multiply the current value in REG1 by ten.
1241H	Save the value in register pair AF on the stack.
1242H - 1243H	Jump.

1244H	Get the value from the stack and put it in register pair AF.
1245H - 1247H	Go divide the current value in REG1 by ten.
1248H	Save the value in register pair AF on the stack.
1249H - 124BH	Go check to see if the current value in REG1 is in range.
124CH	Get the value from the stack and put it in register pair AF.
124DH	Get the value from the stack and put it in register pair DE.
124EH	Return.
124FH	Go check the current value of the number type flag.
1250H - 1252H	Jump if the current value in REG1 is double precision.
1253H - 1255H	Load register pair BC with the exponent and the MSB of a single precision constant.
1256H - 1258H	Load register pair DE with the NMSB and the LSB of a single precision constant. Register pairs BC and DE are now equal to a single precision constant of 999,999.5.
1259H - 125BH	Go compare the single precision constant in register pairs BC and DE to the single precision value in REG1.
125CH - 125DH	Jump.
125EH - 1260H	Load register pair DE with the starting address of a double precision constant equal to 1E16H.
1261H - 1263H	Go compare the double precision constant pointed to by register pair DE to the double precision value in REG1.
1264H	Get the return address from the stack and put it in register pair HL.
1265H - 1267H	Jump if the current value in REG1 isn't in range.
1268H	Jump.
1269H	Check to see if the value in register A is equal to zero.
126AH	Return if the value in register A is equal to zero.
126BH	Decrement the value in register A.

126CH - 126DH	Save a zero character at the location of the input buffer pointer in register pair HL.
126EH	Bump the input buffer pointer in register pair HL.
126FH - 1270H	Jump.
1271H - 1272H	Jump if not done.
1273H	Return if done.
1274H - 1276H	Go put decimal point and commas into the input buffer.
1277H - 1278H	Save a zero character at the location of the input buffer pointer in register pair HL.
1279H	Bump the input buffer pointer in register pair HL.
127AH	Decrement the counter in register A.
127BH - 127CH	Jump.
127DH	Load register A with the value in register E.
127EH	Add the number of digits in register D to the value in register A.
127FH	Bump the adjusted value in register A.
1280H	Load register B with the adjusted value in register A.
1281H	Bump the value in register A.
1282H - 1283H	Subtract three from the adjusted value in register A.
1284H - 1285H	Loop till the value in register A is divided by three.
1286H - 1287H	Adjust the value in register A for the comma count.
1288H	Load register C with the comma count in register A.
1289H - 128BH	Load register A with the value of the edit flag.
128CH - 128DH	Check to see if commas are requested.
128EH	Return if commas are requested.
128FH	Zero the comma counter in register C.
1290H	Return.
1291H	Decrement the decimal point counter in register B.
1292H - 1293H	Jump if the decimal point position hasn't been reached.

1294H - 1295H	Save a decimal point at the location of the input buffer pointer in register pair HL.
1296H - 1298H	Save the address of the decimal point position in register pair HL.
1299H	Bump the input buffer pointer in register pair HL.
129AH	Zero the comma counter in register C.
129BH	Return.
129CH	Decrement the comma counter in register C.
129DH	Return if this location doesn't need a comma.
129EH - 129FH	Save a comma at the location of the input buffer pointer in register pair HL.
12A0H	Bump the input buffer pointer in register pair HL.
12A1H - 12A2H	Set the comma counter in register C to three.
12A3H	Return.
12A4H	Save the value in register pair DE on the stack.
12A5H	Go check the current value of the number type flag.
12A6H - 12A8H	Jump if the current value in REG1 is single precision.
12A9H	Save the value in register pair BC on the stack.
12AAH	Save the value in register pair HL on the stack.
12ABH - 12ADH	Go move the double precision value in REG1 to REG2.
12AEH - 12B0H	Load register pair HL with the starting address of a double precision constant equal to 0.5.
12B1H - 12B3H	Go move the double precision value pointed to by register pair HL to REG1.
12B4H - 12B6H	Go add the double precision value in REG2 to the double precision value in REG1. Return with the double precision result in REG1.
12B7H	Zero register A.
12B8H - 12BAH	Go adjust the double precision result in REG1.
12BBH	Get the value from the stack and put it in register pair HL.

12BCH	Get the value from the stack and put it in register pair BC.
12BDH - 12BFH	Load register pair DE with the starting address of a series of double precision constants for the binary to ASCII conversion.
12C0H - 12C1H	Load register A with the number of times to divide the double precision value in REG1.
12C2H - 12C4H	Go put a comma or a decimal point into the input buffer if necessary.
12C5H	Save the value in register pair BC on the stack.
12C6H	Save the value in register pair AF on the stack.
12C7H	Save the value in register pair HL on the stack.
12C8H	Save the value in register pair DE on the stack.
12C9H - 12CAH	Load register B with the ASCII value for a zero character minus one.
12CBH	Bump the ASCII value for the digit in register B.
12CCH	Get the value from the stack and put it in register pair HL.
12CDH	Save the value in register pair HL on the stack.
12CEH - 12D0H	Go subtract the double precision value pointed to by register pair HL from the double precision value in REG1.
12D1H - 12D2H	Loop till the value of the digit is found.
12D3H	Get the value from the stack and put it in register pair HL.
12D4H - 12D6H	Go add the double precision value pointed to by register pair HL to the double precision remainder in REG1. Return with the correct remainder in REG1.
12D7H	Load register pair DE with the starting address for the next double precision value in register pair HL.
12D8H	Get the value from the stack and put it in register pair HL.
12D9H	Save the ASCII value for the digit in register B at the location of the input buffer pointer in register pair HL.
12DAH	Bump the input buffer pointer in register pair HL.

12DBH	Get the value from the stack and put it in register pair AF.
12DCH	Get the value from the stack and put it in register pair BC.
12DDH	Decrement the counter value in register A.
12DEH - 12DFH	Loop till the ASCII string has been figured.
12E0H	Save the value in register pair BC on the stack.
12E1H	Save the value in register pair HL on the stack.
12E2H - 12E4H	Load register pair HL with the starting address for REG1.
12E5H - 12E7H	Go move the value pointed to by register pair HL into REG1 as a single precision value.
12E8H - 12E9H	Jump.
12EAH	Save the value in register pair BC on the stack.
12EBH	Save the value in register pair HL on the stack.
12ECH - 12EEH	Go add a single precision value of 0.5 to the single precision value in REG1.
12EFH	Bump the value in register A.
12F0H - 12F2H	Go convert the single precision value in REG1 to an integer.
12F3H - 12F5H	Go move the single precision value in register pairs BC and DE into REG1.
12F6H	Get the value from the stack and put it in register pair HL.
12F7H	Get the value from the stack and put it in register pair BC.
12F8H	Zero register A.
12F9H - 12FBH	Load register pair DE with the starting address for a series of integer values.
12FCH	Complement the Carry flag.
12FDH - 12FFH	Go put a decimal point or a comma into the input buffer if necessary.
1300H	Save the value in register pair BC on the stack.
1301H	Save the value in register pair AF on the stack.

1302H	Save the value in register pair HL on the stack.
1303H	Save the value in register pair DE on the stack.
1304H - 1306H	Go move the single precision value in REG1 into register pairs BC and DE.
1307H	Get the value from the stack and put it in register pair HL.
1308H - 1309H	Load register B with the ASCII value for a zero character minus one.
130AH	Bump the ASCII value from the digit in register B.
130BH	Load register A with the LSB of the single precision value in register E.
130CH	Subtract the value at the location of the memory pointer in register pair HL from the value of the LSB of the single precision value in register A.
130DH	Load register E with the adjusted LSB of the single precision value in register A.
130EH	Bump the value of the memory pointer in register pair HL.
130FH	Load register A with the NMSB of the single precision value in register D.
1310H	Subtract the value at the location of the memory pointer in register pair HL from the value of the NMSB of the single precision value in register A.
1311H	Load register D with the adjusted NMSB of the single precision value in register A.
1312H	Bump the value of the memory pointer in register pair HL.
1313H	Load register A with the MSB of the single precision value in register C.
1314H	Subtract the value at the location of the memory pointer in register pair HL from the value of the MSB of the single precision value in register A.
1315H	Load register C with the adjusted MSB of the single precision value in register A.
1316H	Decrement the value of the memory pointer in register pair HL.
1317H	Decrement the value of the memory pointer in register pair HL.

1318H - 1319H	Loop until the ASCII value for the digit has been figured.
131AH - 131CH	Go add the value at the location of the memory pointer in register pair HL to the value in register pairs BC and DE.
131DH	Bump the value of the memory pointer in register pair HL.
131EH - 1320H	Go save the single precision remainder in register pairs BC and DE.
1321H	Load register pair DE with the address of the next value to divide the current value in REG1 by.
1322H	Get the value from the stack and put it in register pair HL.
1323H	Save the ASCII value for the digit in register B at the location of the input buffer pointer in register pair HL.
1324H	Bump the value of the input buffer pointer in register pair HL.
1325H	Get the value from the stack and put it in register pair AF.
1326H	Get the value from the stack and put it in register pair BC.
1327H - 1328H	Loop till the integer portion is found.
1329H	Bump the value of the memory pointer in register pair DE.
132AH	Bump the value of the memory pointer in register pair DE.
132BH - 132CH	Load register A with the number of digits for the ASCII string to be figured.
132DH - 132EH	Jump.
132FH	Save the value in register pair DE on the stack.
1330H - 1332H	Load register pair DE with the starting address of the integer value for figuring the ASCII string.
1333H - 1334H	Load register A with the number of digits for the ASCII string to be figured.
1335H - 1337H	Go put a decimal point or a comma into the input buffer if necessary.
1338H	Save the value in register pair BC on the stack.

1339H	Save the value in register pair AF on the stack.
133AH	Save the value in register pair HL on the stack.
133BH	Load register pair HL with the value of the memory pointer in register pair DE.
133CH	Load register C with the LSB for the integer value at the location of the memory pointer in register pair HL.
133DH	Bump the value of the memory pointer in register pair HL.
133EH	Load register B with the MSB for the integer value at the location of the memory pointer in register pair HL.
133FH	Save the integer value in register pair BC on the stack.
1340H	Bump the value of the memory pointer in register pair HL.
1341H	Exchange the integer value on the stack with the value of the memory pointer in register pair HL.
1342H	Load register pair DE with the integer value in register pair HL.
1343H - 1345H	Load register pair HL with the integer value in REG1.
1346H - 1347H	Load register B with the ASCII value for a zero character minus one.
1348H	Bump the ASCII value for the digit in register B.
1349H	Load register A with the LSB of the integer value in register L.
134AH	Subtract the value of the LSB of the integer value in register E from the value of the LSB of the integer value in register A.
134BH	Load register L with the adjusted value of the LSB of the integer value in register A.
134CH	Load register A with the value of the MSB of the integer value in register H.
134DH	Subtract the MSB of the integer value in register D from the value of the MSB of the integer value in register A.
134EH	Load register H with the adjusted value of the MSB of the integer value in register A.

134FH - 1350H	Loop till the ASCII value for the digit has been figured.
1351H	Add the integer value in register pair DE to the integer value in register pair HL.
1352H - 1354H	Save the integer remainder in register pair HL in REG1.
1355H	Get the value from the stack and put it in register pair DE.
1356H	Get the value from the stack and put it in register pair HL.
1357H	Save the ASCII value for the digit in register B at the location of the input buffer pointer in register pair HL.
1358H	Bump the value of the input buffer pointer in register pair HL.
1359H	Get the value from the stack and put it in register pair AF.
135AH	Get the value from the stack and put it in register pair BC.
135BH	Decrement the value of the counter in register A.
135CH - 135DH	Loop till all of the digits have been figured.
135EH - 1360H	Go put a decimal point or comma into the input buffer if necessary.
1361H	Save the value in register A at the location of the input buffer pointer in register pair HL.
1362H	Get the value from the stack and put it in register pair DE.
1363H	Return.
1364H - 136BH	**DOUBLE PRECISION CONSTANT STORAGE LOCATION**
1364H - 136BH	A double precision constant equal to 1E10 is stored here.
136CH - 1373H	**DOUBLE PRECISION CONSTANT STORAGE LOCATION**
136CH - 1373H	A double precision constant equal to 9.9921E14 is stored here.
1374H - 137BH	**DOUBLE PRECISION CONSTANT STORAGE LOCATION**

1374H - 137BH	A double precision constant equal to 1E16 is stored here.
137CH - 1383H	**DOUBLE PRECISION CONSTANT STORAGE LOCATION**
137CH - 1383H	A double precision constant equal to 0.5 is stored here.
1384H - 138BH	**DOUBLE PRECISION CONSTANT STORAGE LOCATION**
1384H - 138BH	A double precision constant equal to 1E16 is stored here.
138CH - 13D1H	**DOUBLE PRECISION INTEGER CONSTANT STORAGE LOCATION**
13D2H - 13D9H	**SINGLE PRECISION INTEGER CONSTANT STORAGE LOCATION**
13DAH - 13E0H	**INTEGER CONSTANT STORAGE LOCATION**
13E1H - 13E6H	**LEVEL II BASIC MATH ROUTINE**
13E2H - 13E4H	Load register pair HL with the return address.
13E5H	Exchange the value of the jump address on the stack with the value of the return address in register pair HL.
13E6H	Jump to the address in register pair HL.
13E7H - 13F1H	**LEVEL II BASIC SQUARE ROOT ROUTINE**
13E7H - 13E9H	Go move the current value in REG1 on to the stack.
13EAH - 13ECH	Load register pair HL with the starting address of a single precision constant equal to 0.5.
13EDH - 13EFH	Go move the single precision constant pointed to by register pair HL into REG1.
13F0H - 13F1H	Jump.
13F2H - 1478H	**LEVEL II BASIC X ↑ Y ROUTINE**
13F2H - 13F4H	Go convert the value in REG1 to single precision.
13F5H	Get the exponent and the MSB of the single precision value from the stack and put it in register pair BC.
13F6H	Get the NMSB and the LSB of the single precision value from the stack and put it in register pair DE.

13F7H - 13F9H	Go check the sign for the single precision value in REG1.
13FAH	Load register A with the value of the exponent of the single precision value in register B.
13FBH - 13FCH	Jump if the single precision value in REG1 is equal to zero.
13FDH - 13FFH	Jump if the single precision value in REG1 is positive.
1400H	Check to see if the single precision value in register pairs BC and DE is equal to zero.
1401H - 1403H	Go to the Level II BASIC error routine and display a /0 ERROR message if the single precision value in REG1 is negative and the single precision value in register pairs BC and DE is equal to zero.
1404H	Check to see if the single precision value in register pairs BC and DE is equal to zero.
1405H - 1407H	Jump if the single precision value in register pairs BC and DE is equal to zero.
1408H	Save the NMSB and the LSB of the single precision value in register pair DE on the stack.
1409H	Save the exponent and the MSB of the single precision value in register pair BC on the stack.
140AH	Load register A with the value of the MSB of the single precision value in register C.
140BH - 140CH	Mask the MSB of the single precision value in register A.
140DH - 140FH	Go move the single precision value in REG1 into register pairs BC and DE.
1410H - 1412H	Jump if the single precision value on the stack is positive.
1413H	Save the NMSB and the LSB of the single precision value in register pair DE on the stack.
1414H	Save the exponent and the LSB of the single precision value in register pair BC on the stack.
1415H - 1417H	Go figure the integer portion of the single precision value in REG1.
1418H	Get the exponent and the MSB of the single preci-

	sion value from the stack and put it in register pair BC.
1419H	Get the NMSB and the LSB of the single precision value from the stack and put it in register pair DE.
141AH	Save the value in register pair AF on the stack.
141BH - 141DH	Go compare the single precision value in register pairs BC and DE to the single precision value in REG1.
141EH	Get the value from the stack and put it in register pair HL.
141FH	Load register A with the value of the exponent in register H.
1420H	Put bit zero of the exponent in register A into the Carry flag.
1421H	Get the value from the stack and put it in register pair HL.
1422H - 1424H	Save the value in register pair HL as the exponent and the MSB of the single precision value in REG1.
1425H	Get the value from the stack and put it in register pair HL.
1426H - 1428H	Save the value in register pair HL as the NMSB and the LSB of the single precision value in REG1.
1429H - 142BH	Jump if the exponent is an odd number.
142CH - 142EH	Go invert the value of the exponent.
142FH	Save the NMSB and the LSB of the single precision value in register pair DE on the stack.
1430H	Save the exponent and the MSB of the single precision value in register pair BC on the stack.
1431H - 1433H	Go figure the LOG for the current value in REG1.
1434H	Get the exponent and the MSB of the single precision value from the stack and put it in register pair BC.
1435H	Get the NMSB and the LSB of the single precision from the stack and put it in register pair DE.
1436H - 1438H	Go multiply the single precision value in register pairs BC and DE by the current single precision value in REG1.

1439H - 143BH	Go move the single precision value in REG1 on to the stack.
143CH - 143EH	Load register pair BC with the exponent and MSB of a single precision constant.
143FH - 1441H	Load register pair DE with the NMSB and the LSB of a single precision constant. Register pairs BC and DE are now equal to a single precision constant of 1.442695.
1442H - 1444H	Go multiply the single precision value in REG1 by the single precision constant in register pairs BC and DE.
1445H - 1447H	Load register A with the value of the exponent for the single precision value in REG1.
1448H - 1449H	Check to see if the integer portion of the single precision value in REG1 uses more than 7 bits of precision.
144AH - 144CH	Jump if the single precision value in REG1 uses more than 7 bits of precision for it's integer portion.
144DH - 144FH	Go get the integer portion of the value in REG1 and return with it in register A.
1450H - 1451H	Adjust the value in register A.
1452H - 1453H	Adjust the value in register A.
1454H - 1456H	Go check the exponent for the single precision value in REG1 if the value in register A has overflowed.
1457H	Save the value in register pair AF on the stack.
1458H - 145AH	Load register pair HL with the starting address for a single precision constant equal to 1.0.
145BH - 145DH	Go add the single precision constant pointed to by register pair HL to the current value in REG1.
145EH - 1460H	Go multiply the single precision value in REG1 by 0.693147.
1461H	Get the value from the stack and put it in register pair AF.
1462H	Get the exponent and the MSB of the single precision value from the stack and put it in register pair BC.
1463H	Get the NMSB and the LSB of the single precision value from the stack and put it in register pair DE.

1464H	Save the value in register pair AF on the stack.
1465H - 1467H	Go subtract the single precision value in REG1 from the single precision value in register pairs BC and DE. Return with the result in REG1.
1468H - 146AH	Go make the current single precision value in REG1 positive.
146BH - 146DH	Load register pair HL with the starting address for a series of computations.
146EH - 1470H	Go do the series of computations.
1471H - 1473H	Load register pair DE with zero.
1474H	Get the value from the stack and put it in register pair BC.
1475H	Load register C with zero.
1476H - 1478H	Jump.
1479H - 1499H	**SINGLE PRECISION CONSTANT STORAGE LOCATION**
1479H	The number of single precision constants which follow is stored here.
147AH - 147DH	A single precision constant equal to −0.00014171607 is stored here.
147EH - 1481H	A single precision constant equal to 0.00132988204 is stored here.
1482H - 1485H	A single precision constant equal to −0.00830136052 is stored here.
1486H - 1489H	A single precision constant equal to 0.04165735095 is stored here.
148AH - 148DH	A single precision constant equal to −0.16666531543 is stored here.
148EH - 1491H	A single precision constant equal to 0.49999996981 is stored here.
1492H - 1495H	A single precision constant equal to −1.0 is stored here.
1496H - 1499H	A single precision constant equal to 1.0 is stored here.
149AH - 14C8H	**LEVEL II BASIC MATH ROUTINE**
149AH - 149CH	Go move the single precision value in REG1 on to the stack.

149DH - 149FH	Load register pair DE with the return address.
14A0H	Save the return address in register pair DE on the stack.
14A1H	Save the memory pointer in register pair HL on the stack.
14A2H - 14A4H	Go move the single precision value in REG1 into register pairs BC and DE.
14A5H - 14A7H	Go square the single precision value in REG1 into register pairs BC and DE.
14A8H	Get the value of the memory pointer from the stack and put it in register pair HL.
14A9H - 14ABH	Go move the single precision value in REG1 on to the stack.
14ACH	Load register A with the number of values to be figured at the location of the memory pointer in register pair HL.
14ADH	Bump the value of the memory pointer in register pair HL.
14AEH - 14B0H	Go move the single precision value at the location of the memory pointer in register pair HL into REG1.
14B2H	Get the number of values to be figured from the stack and put it in register A.
14B3H	Get the exponent and the MSB of the single precision value from the stack and put it in register pair BC.
14B4H	Get the NMSB and the LSB of the single precision value from the stack and put it in register pair DE.
14B5H	Decrement the value of the counter for the value to be figured in register A.
14B6H	Return if the series of computations has been completed.
14B7H	Save the NMSB and the LSB of the single precision value in register pair DE on the stack.
14B8H	Save the exponent and the MSB of the single precision value in register pair BC on the stack.
14B9H	Save the value of the counter in register A on the stack.
14BAH	Save the value of the memory pointer in register pair HL on the stack.

14BBH - 14BDH	Go multiply the single precision value in register pairs BC and DE by the single precision value in REG1.
14BEH	Get the value of the memory pointer from the stack and put it in register pair HL.
14BFH - 14C1H	Go put the single precision value at the location of the memory pointer in register pair HL in register pairs BC and DE.
14C2H	Save the value of the memory pointer in register pair HL on the stack.
14C3H - 14C5H	Go add the single precision value in REG1 to the single precision value in register pairs BC and DE. Return with the result in REG1.
14C6H	Get the value of the memory pointer from the stack and put it in register pair HL.
14C7H - 14C8H	Jump.
14C9H - 1540H	**LEVEL II BASIC RND ROUTINE**
14C9H - 14CBH	Go convert the current value in REG1 to an integer and return with the integer result in register pair HL.
14CCH	Load register A with the value of the MSB for the integer value in register H.
14CDH	Check to see if the integer value in register pair HL is negative.
14CEH - 14D0H	Got to the Level II BASIC error routine and display a FC ERROR message if the integer value in register pair HL is negative.
14D1H	Check to see if the integer value in register pair HL is equal to zero.
14D2H - 14D4H	Jump if the integer value in register pair HL is equal to zero.
14D5H	Save the integer value in register pair HL on the stack.
14D6H - 14D8H	Go get a RND(0) value and return with the single precision result in REG1.
14D9H - 14DBH	Go move the single precision value in REG1 into register pairs BC and DE.
14DCH	Load register pair HL with the NMSB and the LSB of the single precision value in register pair DE.

14DDH		Exchange the integer value on the stack with NMSB and the LSB of the single precision value in register pair HL.
14DEH		Save the exponent and the MSB of the single precision value in register pair BC on the stack.
14DFH	- 14E1H	Go convert the integer value in register pair HL to single precision and return with the result in REG1.
14E2H		Get the exponent and the MSB of the single precision value from the stack and put it in register pair BC.
14E3H		Get the NMSB and the LSB of the single precision value from the stack and put it in register pair DE.
14E4H	- 14E6H	Go multiply the single precision value in register pairs BC and DE by the single precision value in REG1. Return with the single precision result in REG1.
14E7H	- 14E9H	Load register pair HL with the starting address of a single precision constant equal to 1.0.
14EAH	- 14ECH	Go add the single precision constant pointed to by register pair HL to the single precision value in REG1. Return with the single precision result in REG1.
14EDH	- 14EFH	Jump.
14F0H	- 14F2H	Load register pair HL with the starting address for a table used for figuring random numbers.
14F3H		Save the value of the memory pointer in register pair HL on the stack.
14F4H	- 14F6H	Load register pair DE with zero.
14F7H		Load register C with zero.
14F8H	- 14F9H	Load register H with the counter value.
14FAH	- 14FBH	Load register L with a counter value.
14FCH		Exchange the counters in register pair HL with the NMSB and the LSB of the random number in register pair DE.
14FDH		Multiply the NMSB and the LSB of the random number in register pair HL by two.
14FEH		Exchange the NMSB and the LSB of the random number in register pair HL with the counters in register pair DE.

14FFH	Load register A with the MSB of the random number in register C.
1500H	Multiply the MSB of the random number in register A by two.
1501H	Load register C with the adjusted MSB of the random number in register A.
1502H	Exchange the counter values in register pair HL with the value of the memory pointer on the stack.
1503H	Load register A with the value at the location of the memory pointer in register pair HL.
1504H	Multiply the value in register A by two.
1505H	Save the adjusted value in register A at the location of the memory pointer in register pair HL.
1506H	Exchange the memory pointer in register pair HL with the counter values on the stack.
1507H - 1509H	Jump if the table value hasn't overflowed.
150AH	Save the counter values in register pair HL on the stack.
150BH - 150DH	Load register pair HL with the NMSB and the LSB of the random number seed.
150EH	Add the NMSB and the LSB of the random number in register pair DE to the NMSB and the LSB of the random number seed in register pair HL.
150FH	Load register pair DE with the adjusted NMSB and LSB of the random number in register pair HL.
1510H - 1512H	Load register A with the MSB of the random number seed.
1513H	Add the MSB of the random number in register C to the MSB of the random number seed in register A.
1514H	Load register C with the adjusted MSB of the random number in register A.
1515H	Get the counter values from the stack and put it in register pair HL.
1516H	Decrement the counter in register L.
1517H - 1519H	Loop till the above has been done eight times.
151AH	Exchange the counter values in register pair HL with the memory pointer on the stack.

151BH	Bump the value of the memory pointer in register pair HL.
151CH	Exchange the value of the memory pointer in register pair HL with the counter value on the stack.
151DH	Decrement the counter value in register H.
151EH - 1520H	Loop till the random number has been figured.
1521H	Get the value from the stack and put it in register pair HL.
1522H - 1524H	Load register pair HL with the value to reseed the random number seed.
1525H	Add the value in register pair HL to the NMSB and the LSB of the random number in register pair DE.
1526H - 1528H	Save the adjusted value in register pair HL as the NMSB and the LSB of the random number seed.
1529H - 152BH	Go set the current number type to single precision.
152CH - 152DH	Load register A with a value to reseed the random number seed.
152EH	Add the MSB of the random number in register C to the value in register A.
152FH - 1531H	Save the adjusted value in register A as the MSB of the random number seed.
1532H	Load register pair DE with the NMSB and the LSB of the random number in register pair HL.
1533H - 1534H	Load register B with a value for the sign and the exponent.
1535H - 1537H	Load register pair HL with the address for the sign value storage location.
1538H	Save the sign result in register B at the location of the memory pointer in register pair HL.
1539H	Decrement the value of the memory pointer in register pair HL.
153AH	Save the value of the exponent in register B at the location of the memory pointer in register pair HL.
153BH	Load register C with the value of the MSB for the single precision random number in register A.
153CH - 153DH	Zero the value of the exponent in register B.
153EH - 1540H	Jump.

1541H - 1546H **LEVEL II BASIC COS ROUTINE**

1541H - 1543H Load register pair HL with the starting address of a single precision constant equal to 1.57079637029.

1544H - 1546H Go add the single precision constant pointed to by register pair HL to the single precision value in REG1.

1547H - 158AH **LEVEL II BASIC SIN ROUTINE**

1547H - 1549H Go move the current single precision value in REG1 on to the stack.

154AH - 154CH Load register pair BC with the exponent and the MSB of a single precision constant.

154DH - 154FH Load register pair DE with the NMSB and the LSB of a single precision constant. Register pairs BC and DE now hold a single precision constant equal to 6.2831855.

1550H - 1552H Go move the single precision value in register pairs BC and DE into REG1.

1553H Get the exponent and the MSB of the single precision value from the stack and put it in register pair BC.

1554H Get the NMSB and the LSB of the single precision value from the stack and put it in register pair DE.

1555H - 1557H Go divide the single precision value in register pairs BC and DE by the single precision value in REG1. Return with the single precision result in REG1.

1558H - 155AH Go move the single precision result in REG1 onto the stack.

155BH - 155DH Go figure the integer portion for the single precision value in REG1.

155EH Get the exponent and the MSB of the single precision value from the stack and put it in register pair BC.

155FH Get the NMSB and the LSB of the single precision value from the stack and put it in register pair DE.

1560H - 1562H Go subtract the single precision value in REG1 from the single precision value in register pairs BC and DE. Return with the result in REG1.

1563H - 1565H Load register pair HL with the starting address of a single precision constant equal to 0.25.

1566H - 1568H Go subtract the single precision value in REG1 from

	the single precision constant pointed to by register pair HL. Return with the result in REG1.
1569H - 156BH	Go check the sign for the single precision value in REG1.
156CH	Set the Carry flag.
156DH - 156FH	Jump if the single precision value in REG1 is positive.
1570H - 1572H	Go add .5 to the single precision value in REG1. Return with the result in REG1.
1573H - 1575H	Go check the sign for the single precision value in REG1.
1576H	Test the value of the sign test in register A.
1577H	Save the value in register pair AF on the stack.
1578H - 157AH	Go adjust the sign for the single precision value in REG1 if necessary.
157BH - 157DH	Load register pair HL with the starting address of a single precision constant equal to 0.25.
157EH - 1580H	Go add the single precision constant pointed to by register pair HL to the single precision value in REG1. Return with the result in REG1.
1581H	Get the value from the stack and put it in register pair AF.
1582H - 1584H	Jump if the Carry flag wasn't set from above.
1585H - 1587H	Load register pair HL with the starting address for a series of single precision values for a set of computations.
1588H - 158AH	Go do the computations.
158BH - 158EH	**SINGLE PRECISION CONSTANT STORAGE LOCATION**
158BH - 158EH	A single precision constant equal to 1.57079637029 is stored here.
158FH - 1592H	**SINGLE PRECISION CONSTANT STORAGE LOCATION**
158FH - 1592H	A single precision constant equal to 0.25 is stored here.
1593H - 15A7H	**SINGLE PRECISION CONSTANTS STORAGE LOCATION**

1593H	The number of single precision constants which follows is stored here.
1594H - 1597H	A single precision constant equal to 39.7106704708 is stored here.
1598H - 159BH	A single precision constant equal to −76.5749816893 is stored here.
159CH - 159FH	A single precision constant equal to 81.6022338865 is stored here.
15A0H - 15A3H	A single precision constant equal to −41.3416748045 is stored here.
15A4H - 15A7H	A single precision constant equal to 6.28318500497 is stored here.
15A8H - 15BCH	**LEVEL II BASIC TAN ROUTINE**
15A8H - 15AAH	Go move the single precision value in REG1 on to the stack.
15ABH - 15ADH	Go figure the SIN for the single precision value in REG1. Return with the result in REG1.
15AEH	Get the exponent and the MSB for the single precision value on the stack and put it in register pair BC.
15AFH	Get the NMSB and the LSB of the single precision value on the stack and put it in register pair HL.
15B0H - 15B2H	Go move the single precision value in REG1 on to the stack.
15B3H	Load register pair DE with the NMSB and the LSB of the single precision value in register pair HL.
15B4H - 15B6H	Go move the single precision value in register pairs BC and DE into REG1.
15B7H - 15B9H	Go figure the COS for the single precision value in REG1. Return with the result in REG1.
15BAH - 15BCH	Jump.
15BDH - 15E2H	**LEVEL II BASIC ATN ROUTINE**
15BDH - 15BFH	Go check the sign of the single precision value in REG1.
15C0H - 15C2H	If the single precision value in REG1 is negative then put a return address on the stack.
15C3H - 15C5H	Go convert the negative number in REG1 to positive if necessary.

15C6H - 15C8H	Load register A with the exponent of the single precision value in REG1.
15C9H - 15CAH	Check to see if the single precision value in REG1 is less than one.
15CBH - 15CCH	Jump if the single precision value in REG1 is less than one.
15CDH - 15CFH	Load register pair BC with an exponent and a MSB for a single precision value.
15D0H	Zero the NMSB of the single precision value in register D.
15D1H	Zero the LSB of the single precision value in register E.
15D2H - 15D4H	Go divide the single precision value in REG1 into the single precision constant in register pairs BC and DE.
15D5H - 15D7H	Load register pair HL with the return address.
15D8H	Save the value of the return address in register pair HL on the stack.
15D9H - 15DBH	Load register pair HL with the starting address for a series of single precision numbers for a set of computations.
15DCH - 15DEH	Go do the set of computations.
15DFH - 15E1H	Load register pair HL with the starting address of a single precision constant equal to 1.57079637029.
15E2H	Return.
15E3H - 1607H	**SINGLE PRECISION CONSTANTS STORAGE LOCATION**
15E3H	The number of single precision constants which follows is stored here.
15E4H - 15E7H	A single precision constant equal to 0.00286622549 is stored here.
15E8H - 15EBH	A single precision constant equal to −0.01616573699 is stored here.
15ECH - 15EFH	A single precision constant equal to 0.04290961441 is stored here.
15F0H - 15F3H	A single precision constant equal to 0.07528963666 is stored here.

15F4H - 15F7H A single precision constant equal to 0.10656264407 is stored here.

15F8H - 15FBH A single precision constant equal to −0.14208900905 is stored here.

15FCH - 15FFH A single precision constant equal to 0.19993549561 is stored here.

1600H - 1603H A single precision constant equal to −0.33333146561 is stored here.

1604H - 1607H A single precision constant equal to 1.0 is stored here.

1608H - 18C8H **LIST OF BASIC RESERVED WORDS, TOKENS, AND ENTRY LOCATIONS AS FOLLOWS:**

Reserved words, and their tokens, and their entry points.

ABS	D9	0977	AND	D2	25FD
ASC	F6	2AOF	ATN	E4	15BD
AUTO	B7	2008	CDBL	F1	OADB
CHR$(	F7	2A1F	CINT	EF	0A7F
CLEAR	B8	1E7A	CLOAD	B9	2C1F
CLOSE	A6	4185	CLS	84	01C9
CMD	85	4173	CONT	B3	1DE4
COS	E1	1541	CSAVE	BA	2BF5
CSNG	F0	0AB1	CVD	E8	415E
CVI	E6	4152	CVS	E7	4158
DATA	88	1F05	DEF	DD	415B
DEFDBL	9B	1E09	DEFINT	99	1E03
DEFSNG	9A	1E06	DEFSTR	98	1E00
DELETE	B6	2BC6	DIM	8A	2608
EDIT	9D	2E60	ELSE	95	1F07
END	80	1DAE	EOF	E9	4161
ERL	C2	24DD	ERR	C3	24CF
ERROR	9E	1FF4	EXP	EO	1439
FIELD	A3	417C	FIX	F2	0B26
FN	BE	4155	FOR	81	1CA1
FRE	DA	27D4	GET	A4	4174
GOSUB	91	1EB1	GOTO	8D	1EC2
			IF	8F	2039
INKEY$	C9	019D	INP	DB	2AEF
INPUT	89	219A	INSTR	C5	419D
INT	D8	0B37	KILL	AA	4191
LEFT$	F8	2A61	LEN	F3	2A03
LET	8C	1F21	LINE	9C	41A3
LIST	B4	2B2E	LLIST	B5	2B29
LOAD	A7	4188	LOC	EA	4164
LOF	EB	4167	LOG	DF	0809
LPRINT	AF	2067	LSET	AB	4197
MEM	C8	27C9	MERGE	A8	418B
MID$	FA	2A9A	MKD$	EE	4170

MKI$	EC	416A	MKS$	ED	416D
NAME	A9	418E	NEW	BB	1B49
NEXT	87	22B6	NOT	CB	25C4
ON	A1	1FC6	OPEN	A2	4179
OR	D3	25F7	OUT	A0	2AFB
PEEK	E5	2CAA	POINT	C6	0132
POKE	B1	2CB1	POS	DC	27F5
PRINT	B2	206F	PUT	A5	4182
RANDOM	86	01D3	READ	8B	21EF
REM	93	1F07	RESET	82	0138
RESTORE	90	1D91	RESUME	9F	1FAF
RETURN	92	1EDE	RIGHT$	F9	2A91
RND	DE	14C9	RSET	AC	419A
RUN	8E	1EA3	SAVE	AD	41A0
SET	83	0135	SGN	D7	098A
SIN	E2	1547	SQR	CD	13E7
STEP	CC	2B01	STOP	94	1DA9
STR$	F4	2836	STRING$	C4	2A2F
SYSTEM	AE	02B2	TAB(	BC	2137
TAN	E3	15A8	THEN	CA	----
TIME$	C7	4176	TO	BD	----
TROFF	97	1DF8	TRON	96	1DF8
USING	BF	2CBD	USR	C1	27FE
VAL	FF	2AC5	VARPTR	C0	24EB
+	CD	249F	–	CE	2532
*	CF	----	/	D0	----
↑	D1	----	>	D4	----
=	D5	----	<	D6	----
&	26	4194	' 3A 93	FB	----

18C9H - 18F6H **STORAGE LOCATION FOR LEVEL II BASIC ERROR MESSAGES**

18F7H - 1904H **STORAGE LOCATION FOR THE DIVISION ROUTINE**

1095H - 191CH **STORAGE LOCATION FOR VALUES PLACED IN RAM UPON INITIALIZATION.**

191DH - 1923H **MESSAGE STORAGE LOCATION**

191DH - 1923H The Level II BASIC ERROR message is stored here.

1924H - 1928H **MESSAGE STORAGE LOCATION**

1924H - 1928H The Level II BASIC IN message is stored here.

1929H - 192FH **MESSAGE STORAGE LOCATION**

1929H - 192FH The Level II BASIC READY message is stored here.

1930H - 1935H **MESSAGE STORAGE LOCATION**

1930H - 1935H The Level II BASIC BREAK message is stored here.

1936H - 1954H SCAN STACK ROUTINE

1936H - 1938H Load register pair HL with 4.

1939H Add the current value of the stack pointer to the value in register pair HL.

193AH Load register A with the value at the location of the memory pointer in register pair HL.

193BH Bump the value of the memory pointer in register pair HL.

193CH - 193DH Check to see if the value in register A is a FOR token.

193EH Return if the value in register A isn't a FOR token.

193FH Load register C with the LSB of the FOR's variable address.

1940H Bump the value of the memory pointer in register pair HL.

1941H Load register B with the MSB of the FOR's variable address.

1942H Bump the value of the memory pointer in register pair HL.

1943H Save the value in register pair HL on the stack.

1944H Load register L with the LSB of the FOR variable's address in register C.

1945H Load register H with the MSB of the FOR variable's address in register B.

1946H Load register A with the MSB of the variable address in register D.

1947H Check to see if the variable address in register pair DE is equal to zero.

1948H Exchange the variable address in register pair HL with the variable address in register pair DE.

1949H - 194AH Jump if the variable address in register pair DE was equal to zero.

194BH Exchange the variable address in register pair HL with the variable address in register pair DE.

194CH Go check to see if the variable address in register pair HL is the same as the variable address in register pair DE.

194DH - 194FH	Load register pair BC with the value to increment the memory pointer.
1950H	Get the memory pointer from the stack and put it in register pair HL.
1951H	Return if the FOR block has been found.
1952H	Add the value in register pair BC to the value of the memory pointer in register pair HL.
1953H - 1954H	Loop till the FOR block has been located.
1955H - 1962H	**DATA MOVEMENT ROUTINE**
1955H - 1957H	Go make sure there's enough room in memory.
1958H	Save the value of the memory pointer in register pair BC on the stack.
1959H	Exchange the value of the memory pointer in register pair HL with the value of the memory pointer on the stack.
195AH	Get the value of the memory pointer from the stack and put it in register pair BC.
195BH	Check to see if the memory pointer in register pair HL is the same as the memory pointer in register pair DE.
195CH	Load register A with the value at the location of the memory pointer in register pair HL.
195DH	Save the value in register A at the location of the memory pointer in register pair BC.
195EH	Return if the memory pointer in register pair HL is the same as the value of the memory pointer in register pair DE.
195FH	Decrement the value of the memory pointer in register pair BC.
1960H	Decrement the value of the memory pointer in register pair HL.
1961H - 1962H	Loop till done.
1963H - 197DH	**MEMORY CHECK ROUTINE**
1963H	Save the value in register pair HL on the stack.
1964H - 1966H	Load register pair HL with the starting address of free memory.
1967H - 1968H	Load register B with zero.

1969H	Add the value in register pair BC to the value in register pair HL.
196AH	Add the value in register pair BC to the value in register pair HL.
196CH	Save the value in register pair HL on the stack.
196DH - 196EH	Load register A with the LSB of the top of memory.
196FH	Subtract the LSB of the value of the new memory pointer in register L from the value in register A.
1970H	Load register L with the adjusted value in register A.
1971H - 1972H	Load register A with the MSB of the top of memory.
1973H	Subtract the MSB of the new memory pointer in register H from the value in register A.
1974H - 1975H	Jump if out of memory.
1976H	Load register H with the adjusted value in register A.
1977H	Add the value of the stack pointer to the adjusted value in register pair HL.
1978H	Get the value from the stack and put it in register pair HL.
1979H	Return if overflow didn't occur.
197AH - 197BH	Load register E with the OM ERROR code.
197CH - 197DH	Go to the Level II BASIC error routine and display an OM ERROR message.
197EH - 1AF7H	**LEVEL II BASIC COMMAND MODE**
197EH - 1980H	Load register pair HL with the value of the current BASIC line number.
1981H	Load register A with the MSB of the current BASIC line number in register H.
1982H	Combine the LSB of the current BASIC line number in register L with the MSB of the current line number in register A.
1983H	Bump the value of the combined BASIC line number in register A.
1984H - 1985H	Jump if Level II BASIC is still in the command mode.
1986H - 1988H	Load register A with the error flag.

1989H	Check to see if the error flag is set.
198AH - 198BH	Load register E with a NR ERROR code.
198CH - 198DH	Jump if the error flag is set.
198EH - 1990H	Jump.
1991H - 1993H	Load register pair HL with the DATA line number.
1994H - 1996H	Save the DATA line number in register pair HL.
1997H - 1998H	Load register E with a SN ERROR code.
199AH - 199BH	Load register E with a /0 ERROR code.
199DH - 199EH	Load register E with a NF ERROR code.
19A0H - 19A1H	Load register E with a RW ERROR code.
19A2H - 19A4H	Load register pair HL with the value of the current BASIC line number.
19A5H - 19A7H	Save the value of the current BASIC line number in register pair HL.
19A8H - 19AAH	Save the value of the current BASIC line number in register pair HL.
19ABH - 19ADH	Load register pair BC with the return address.
19AEH - 19B0H	Load register pair HL with the value of the stack pointer.
19B1H - 19B3H	Jump.
19B4H	Get the value from the stack and put it in register pair BC.
19B5H	Load register A with the value of the error code in register E.
19B6H	Load register C with the value of the error code in register E.
19B7H - 19B9H	Save the value of the error code in register A.
19BAH - 19BCH	Load register pair HL with the value of the current BASIC program pointer.
19BDH - 19BFH	Save the value of the current BASIC program pointer in register pair HL.
19C0H	Load register pair DE with the value of the current BASIC program pointer in register pair HL.

19C1H - 19C3H	Load register pair HL with the value of the current BASIC line number.
19C4H	Load register A with the MSB of the current BASIC line number in register H.
19C5H	Combine the LSB of the current BASIC line number in register L with the MSB of the current BASIC line number in register A.
19C6H	Bump the value of the combined current BASIC line number.
19C7H - 19C8H	Jump if Level II BASIC is in the command mode.
19C9H - 19CBH	Save the value of the current BASIC line number in register pair HL.
19CCH	Load register pair HL with the value of the current BASIC program pointer in register pair DE.
19CDH - 19CFH	Save the value of the current BASIC program pointer in register pair HL.
19D0H - 19D2H	Load register pair HL with the current ONERROR address.
19D3H	Load register A with the MSB of the ONERROR address in register H.
19D4H	Combine the LSB of the ONERROR address in register L with the MSB of the ONERROR address in register A.
19D5H	Load register pair DE with the ONERROR address in register pair HL.
19D6H - 19D8H	Load register pair HL with the address of the error flag.
19D9H - 19DAH	Jump if there isn't an ONERROR address.
19DBH	Combine the value of the error flag at the location of the memory pointer in register pair HL with the combined value of the ONERROR address in register A.
19DCH - 19DDH	Jump if register A is nonzero.
19DEH	Decrement the value of the error flag at the location of the memory pointer in register pair HL.
19DFH	Load register pair HL with the ONERROR address in register pair DE.
19E0H - 19E2H	Jump.

19E3H	Zero register A.
19E4H	Save the value in register A as the current error flag at the location of the memory pointer in register pair HL.
19E5H	Load register E with the value of the error code in register C.
19E6H - 19E8H	Go display a carriage return on the video display if necessary.
19E9H - 19EBH	Load register pair HL with the starting address for the table of error messages.
19ECH - 19EEH	Go to the DOS link at 41A6H.
19EFH	Load register D with zero.
19F0H - 19F1H	Load register A with a question mark.
19F2H - 19F4H	Go display the question mark in register A.
19F5H	Add the value of the error code in register pair DE to the starting address of the table of error messages in register pair HL.
19F6H	Load register A with the first character of the error message at the location of the table pointer in register pair HL.
19F7H - 19F9H	Go display the first character of the error message in register A.
19FAH	Go set the second character of the error in register A.
19FBH - 19FDH	Go display the second character of the error message in register A.
19FEH - 1A00H	Load register pair HL with the starting address of the READY message.
1A01H	Save the value in register pair HL on the stack.
1A02H - 1A04H	Load register pair HL with the value of the current BASIC line number.
1A05H	Exchange the value of the current BASIC line number in register pair HL with the starting address of the Level II BASIC ERROR message on the stack.
1A06H - 1A08H	Go display the Level II BASIC ERROR message.
1A09H	Get the value of the current BASIC line number from the stack and put it in register pair HL.

1A0AH - 1A0CH	Load register pair DE with 65534.
1A0DH	Go compare the value of the current BASIC line number in register pair HL with the line number in register pair DE.
1A0EH - 1A10H	Jump if the line numbers in register pairs HL and DE are the same.
1A11H	Load register A with the MSB of the current BASIC line number in register H.
1A12H	Combine the LSB of the current BASIC line number in register L with the MSB of the current BASIC line number in register A.
1A13H	Bump the combined value of the current BASIC line number in register A.
1A14H - 1A16H	Go display the current BASIC line number in register pair HL if Level II BASIC isn't in the command mode.
1A18H	Get the value from the stack and put it in register pair BC.
1A19H - 1A1BH	Go set the current output device to the video display.
1A1CH - 1A1EH	Go call the DOS link at 41ACH.
1A1FH - 1A21H	Go turn off the cassette recorder.
1A22H - 1A24H	Go display a carriage return if necessary.
1A25H - 1A27H	Load register pair HL with the starting address of the Level II BASIC READY message.
1A28H - 1A2AH	Go display the Level II BASIC READY message.
1A2BH - 1A2DH	Load register A with the value of the current error code.
1A2EH - 1A2FH	Check to see if the current error code is a SN ERROR code.
1A30H - 1A32H	Go to the EDIT mode if the current error code is a SN ERROR code.
1A33H - 1A35H	Load register pair HL with the command mode line number.
1A36H - 1A38H	Save the line number in register pair HL as the current BASIC line number.
1A39H - 1A3BH	Load register A with the value of the AUTO flag.

1A3CH	Check to see if in the AUTO mode.
1A3DH - 1A3EH	Jump if not in the AUTO mode.
1A3FH - 1A41H	Load register pair HL with the current AUTO line number.
1A42H	Save the current AUTO line number in register pair HL on the stack.
1A43H - 1A45H	Go display the current AUTO line number in register pair HL on the video display.
1A46H	Get the current AUTO line number from the stack and put it in register pair DE.
1A47H	Save the current AUTO line number in register pair DE on the stack.
1A48H - 1A4AH	Go check to see if there is a matching line number in the BASIC program.
1A4BH - 1A4CH	Load register A with an *.
1A4DH - 1A4EH	Jump if there is a matching line number in the BASIC program.
1A4FH - 1A50H	Load register A with a space.
1A51H - 1A53H	Go display the character in register A on the video display.
1A54H - 1A56H	Go get the input.
1A57H	Get the current line number from the stack and put it in register pair DE.
1A58H - 1A59H	Jump if the BREAK key wasn't pressed.
1A5AH	Zero register A.
1A5BH - 1A5DH	Save the value in register A as the current AUTO flag.
1A5EH - 1A5FH	Jump to the normal command mode.
1A60H - 1A62H	Load register pair HL with the value of the AUTO increment.
1A63H	Add the value of the AUTO line number in register pair DE with the AUTO increment value in register pair HL.
1A64H - 1A65H	Jump if AUTO line number has overflowed.
1A66H	Save the current AUTO line number in register pair DE on the stack.

1A67H - 1A69H	Load register pair DE with the maximum BASIC line number.
1A6AH	Go compare the adjusted AUTO line number in register pair HL with the value of the maximum BASIC line number in register pair DE.
1A6BH	Get the current AUTO line number from the stack and put it in register pair DE.
1A6CH - 1A6DH	Jump if the adjusted AUTO line number in register pair HL is too large.
1A6EH - 1A70H	Save the adjusted AUTO line number in register pair HL as the current AUTO line number.
1A71H - 1A72H	Load register A with the AUTO flag value.
1A73H - 1A75H	Jump to the EDIT routine to save the current BASIC line.
1A76H - 1A77H	Load register A with a >.
1A78H - 1A7AH	Go display the Level II BASIC prompt in register A on the video display.
1A7BH - 1A7DH	Go get the input.
1A7EH - 1A80H	Jump if the BREAK key was pressed.
1A81H	Go bump the current input buffer pointer in register pair HL till it points to the first character.
1A82H	Bump the value of the character in register A.
1A83H	Decrement the value of the character in register A.
1A84H - 1A86H	Jump if there wasn't any input.
1A87H	Save the value in register pair AF on the stack.
1A88H - 1A8AH	Go evaluate the line number at the location of the input buffer pointer in register pair HL and return with it's binary value in register pair DE.
1A8BH	Decrement the value of the input buffer pointer in register pair HL.
1A8CH	Load register A with the value of the character at the location of the input buffer pointer in register pair HL.
1A8DH - 1A8EH	Check to see if the character at the location of the input buffer pointer in register A is a space.
1A8FH - 1A90H	Jump if the character at the location of the input buffer pointer in register A is a space.

1A91H	Bump the value of the input buffer pointer in register pair HL.
1A92H	Load register A with the character at the location of the input buffer pointer in register pair HL.
1A93H - 1A94H	Check to see if the character at the location of the input buffer pointer in register A is a space.
1A95H - 1A97H	Go bump the value of the input buffer pointer in register pair HL if necessary.
1A98H	Save the BASIC line number in register pair DE on the stack.
1A99H - 1A9BH	Go tokenize the input.
1A9CH	Get the value of the BASIC line number from the stack.
1A9DH	Get the value from the stack and put it in register pair AF.
1A9EH - 1AA0H	Save the input buffer pointer in register pair HL.
1AA1H - 1AA3H	Go call the DOS line at 41B2H.
1AA4H - 1AA6H	Jump if there wasn't a line number with the input.
1AA7H	Save the BASIC line number in register pair DE on the stack.
1AA8H	Save the length of the tokenized input in register pair BC on the stack.
1AA9H	Zero register A.
1AAAH - 1AACH	Save the value in register A as the input flag.
1AADH	Go bump the value of the input buffer pointer in register pair HL till it points to the next character.
1AAEH	Set the flags.
1AAFH	Save the value in register pair AF on the stack.
1AB0H	Load register pair HL with the value of the BASIC line number in register pair DE.
1AB1H - 1AB3H	Save the value of the line number in register pair HL.
1AB4H	Exchange the value of the input buffer pointer in register pair DE with the value of the BASIC line number in register pair HL.

1AB5H - 1AB7H	Go check to see if there is a matching line number in the BASIC program.
1AB8H	Save the address of the line number in the BASIC program in register pair BC on the stack.
1AB9H - 1ABBH	If there wasn't a matching line number in the BASIC program, go move the closest line number up in memory.
1ABCH	Get the address of the BASIC line number from the stack and put it in register pair DE.
1ABDH	Get the value from the stack and put it in register pair AF.
1ABEH	Save the address of the BASIC line number on the stack.
1ABFH - 1AC0H	Jump if there was a matching line number in the BASIC program.
1AC1H	Get the address of the BASIC line number from the stack and put it in register pair DE.
1AC2H - 1AC4H	Load register pair HL with the end of the BASIC program pointer.
1AC5H	Exchange the length of the tokenized input on the stack with the end of the BASIC program pointer in register pair HL.
1AC6H	Get the end of the BASIC program pointer from the stack and put it in register pair BC.
1AC7H	Add the end of the BASIC program pointer in register pair BC to the value of the length of the tokenized input in register pair HL.
1AC8H	Save the adjusted end of the BASIC program pointer in register pair HL on the stack.
1AC9H - 1ACBH	Go check to see if there is enough room in memory for the new BASIC line.
1ACCH	Get the new end of the BASIC program pointer from the stack and put it in register pair HL.
1ACDH - 1ACFH	Save the new end of the BASIC program pointer in register pair HL.
1AD0H	Load register pair HL with the address of the BASIC line.
1AD1H	Save the MSB of the address of the BASIC line in register H.

1AD2H	Get the value of the BASIC line number from the stack and put it in register pair DE.
1AD3H	Save the value of the memory pointer in register pair HL on the stack.
1AD4H	Bump the value of the memory pointer in register pair HL.
1AD5H	Bump the value of the memory pointer in register pair HL.
1AD6H	Save the LSB of the BASIC line number in register E at the location of the memory pointer in register pair HL.
1AD7H	Bump the value of the memory pointer in register pair HL.
1AD8H	Save the MSB of the BASIC line number in register D at the location of the memory pointer in register pair HL.
1AD9H	Bump the value of the BASIC line number in register pair HL.
1ADAH	Load register pair DE with the value of the memory pointer in register pair HL.
1ADBH - 1ADDH	Load register pair HL with the value of the tokenized input pointer.
1ADEH	Exchange the value of the memory pointer in register pair DE with the value of the tokenized input pointer in register pair HL.
1ADFH	Decrement the value of the tokenized input buffer pointer in register pair DE.
1AE0H	Decrement the value of the tokenized input buffer pointer in register pair DE.
1AE1H	Load register A with the value at the location of the tokenized input buffer pointer in register pair DE.
1AE2H	Save the value in register A at the location of the memory pointer in register pair HL.
1AE3H	Bump the value of the memory pointer in register pair HL.
1AE4H	Bump the value of the tokenized input buffer pointer in register pair DE.
1AE5H	Check to see if the character in register A is an end of the line character.

1AE6H - 1AE7H	Loop till the whole of the new BASIC line has been stored in memory.
1AE8H	Get the value from the stack and put it in register pair DE.
1AE9H - 1AEBH	Go reset the BASIC line pointers.
1AECH - 1AEEH	Go call the DOS link at 41B5H.
1AEFH - 1AF1H	Go reset the current BASIC program pointers.
1AF2H - 1AF4H	Go call the DOS link at 41B8H.
1AF5H - 1AF7H	Go get the input again.
1AF8H - 1B0FH	**LINE POINTERS ROUTINE**
1AF8H - 1AFAH	Load register pair HL with the start of the BASIC program pointer.
1AFBH	Load register pair DE with the start of the BASIC program pointer in register pair HL.
1AFCH	Load register H with the MSB of the memory pointer in register D.
1AFDH	Load register L with the LSB of the memory pointer in register E.
1AFEH	Load register A with the value at the location of the memory pointer in register pair HL.
1AFFH	Bump the value of the memory pointer in register pair HL.
1B00H	Check to see if the character at the location of the memory pointer in register pair HL is an end of the BASIC program character.
1B01H	Return if done.
1B02H	Bump the value of the memory pointer in register pair HL.
1B03H	Bump the value of the memory pointer in register pair HL.
1B04H	Bump the value of the memory pointer in register pair HL.
1B05H	Zero register A.
1B06H	Check to see if the character at the location of the memory pointer in register HL is an end of the BASIC line character.

1B07H	Bump the value of the memory pointer in register pair HL.
1B08H - 1B09H	Loop till the end of the BASIC line character is found.
1B0AH	Exchange the starting address of the current BASIC line in register pair DE with the starting address of the next BASIC line in register pair HL.
1B0BH	Save the LSB of the next BASIC line's starting address in register E at the location of the memory pointer in register pair HL.
1B0CH	Bump the value of the memory pointer in register pair HL.
1B0DH	Save the MSB of the next BASIC line's starting address in register D at the location of the memory pointer in register pair HL.
1B0EH - 1B0FH	Loop till done.
1B10H - 1B48H	**EVALUATE LINE NUMBERS**
1B10H - 1B12H	Load register pair DE with zero.
1B13H	Save the value in register pair DE on the stack.
1B14H - 1B15H	Jump if there aren't any line numbers to be evaluated.
1B16H	Get the value from the stack and put it in register pair DE.
1B17H - 1B19H	Go evaluate the line number at the location of the current BASIC program pointer in register pair HL and return with the line number's binary value in register pair DE.
1B1AH	Save the first line number's value in register pair DE on the stack.
1B1BH - 1B1CH	Jump if there isn't a second line number to be evaluated.
1B1DH	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a – token.
1B1EH	The character to be checked for is stored here.
1B1FH - 1B21H	Load register pair DE with the default second line number.
1B22H - 1B24H	Go evaluate the second line number at the location of

	the current BASIC program pointer in register pair HL and return with the line number's binary value in register pair DE.
1B25H - 1B27H	Go to the Level II BASIC error routine and display a SN ERROR message if the data which followed the – token wasn't a line number.
1B28H	Load register pair HL with the value of the second line number in register pair DE and load register pair DE with the value of the current BASIC program pointer in register pair HL.
1B29H	Get the value of the first line number from the stack and put it in register pair DE.
1B2AH	Exchange the return address on the stack with the value of the second line number in register pair HL.
1B2BH	Save the value of the return address in register pair HL on the stack.
1B2CH - 1B2EH	Load register pair HL with the value of the start of the BASIC program pointer.
1B2FH	Load register B with the value of the MSB of the memory pointer in register H.
1B30H	Load register C with the LSB of the memory pointer in register L.
1B31H	Load register A with the value at the location of the memory pointer in register pair HL.
1B32H	Bump the value of the memory pointer in register pair HL.
1B33H	Combine the value at the location of the memory pointer in register pair HL with the value in register A.
1B34H	Decrement the value of the memory pointer in register pair HL.
1B35H	Return if this is the end of the BASIC program.
1B36H	Bump the value of the memory pointer in register pair HL.
1B37H	Bump the value of the memory pointer in register pair HL.
1B38H	Load register A with the LSB of the current BASIC line number at the location of the memory pointer in register pair HL.

1B39H	Bump the value of the memory pointer in register pair HL.
1B3AH	Load register H with the MSB of the current BASIC line number at the location of the memory pointer in register pair HL.
1B3BH	Load register L with the LSB of the current BASIC line number in register A.
1B3CH	Go compare the value of the first line number in register pair DE with the value of the current BASIC line number in register pair HL.
1B3DH	Load register H with the MSB of the memory pointer in register B.
1B3EH	Load register L with the LSB of the memory pointer in register C.
1B3FH	Load register A with the value at the location of the memory pointer in register pair HL.
1B40H	Bump the value of the memory pointer in register pair HL.
1B41H	Load register H with the MSB of the next BASIC line pointer at the location of the memory pointer in register pair HL.
1B42H	Load register L with the LSB of the next BASIC line pointer in register A.
1B43H	Complement the value of the Carry flag.
1B44H	Return if the first line number in register pair DE is the same as the current BASIC line number.
1B45H	Complement the value of the Carry flag.
1B46H	Return if the first line number in register pair DE is less than the current BASIC line number.
1B47H - 1B48H	Loop till the location of the line number has been found in the BASIC program.
1B49H - 1B5CH	**LEVEL II BASIC NEW ROUTINE**
1B49H	Go to the Level II BASIC error routine and display a SN ERROR message if there is any input following the NEW token.
1B4AH - 1B4CH	Go clear the screen.
1B4DH - 1B4FH	Load register pair HL with the start of the BASIC program.

1B50H - 1B52H	Go TROFF.
1B53H - 1B55H	Reset the AUTO flag.
1B56H	Save a zero at the location of the memory pointer in register pair HL.
1B57H	Bump the value of the memory pointer in register pair HL.
1B58H	Save a zero at the location of the memory pointer in register pair HL.
1B59H	Bump the value of the memory pointer in register pair HL.
1B5AH - 1B5CH	Save the value in register pair HL as the new simple variables pointer.
1B5DH - 1BB2H	**LEVEL II BASIC RUN ROUTINE**
1B5DH - 1B5FH	Load register pair HL with the start of the BASIC program pointer.
1B60H	Decrement the value in register pair HL.
1B61H - 1B63H	Save the adjusted value in register pair HL.
1B64H - 1B65H	Load register B with the number of variable names to be initialized.
1B66H - 1B68H	Load register pair HL with the starting address of the variable declaration table.
1B69H - 1B6AH	Set the variable at the location of the memory pointer in register pair HL to a single precision variable.
1B6BH	Bump the value of the memory pointer in register pair HL.
1B6CH - 1B6DH	Loop till all 26 variables have been set to single precision.
1B6EH	Zero register A.
1B6FH - 1B71H	Save the value in register A as the current value of the RESUME flag.
1B72H	Zero register L.
1B73H	Zero register H.
1B74H - 1B76H	Save the value in register pair HL as the current ONERROR address.

1B77H - 1B79H	Save the value in register pair HL as the current BREAK address.
1B7AH - 1B7CH	Load register pair HL with the top of the memory pointer.
1B7DH - 1B7FH	Save the value in register pair HL as the next available address in the string space pointer.
1B80H - 1B82H	Go do a RESTORE.
1B83H - 1B85H	Load register pair HL with the end of the BASIC program pointer.
1B86H - 1B88H	Save the value in register pair HL as the new simple variables pointer.
1B89H - 1B8BH	Save the value in register pair HL as the new array variables pointer.
1B8CH - 1B8EH	Go call the DOS line at 41BBH.
1B8FH	Get the return address from the stack.
1B90H - 1B92H	Load register pair HL with the start of string space pointer.
1B93H	Decrement the value of the memory pointer in register pair HL.
1B94H	Decrement the value of the memory pointer in register pair HL.
1B95H - 1B97H	Save the value in register pair HL as the stack pointer.
1B98H	Bump the value of the memory pointer in register pair HL.
1B99H	Bump the value of the memory pointer in register pair HL.
1B9AH	Load the stack pointer with the value of the memory pointer in register pair HL.
1B9BH - 1B9DH	Load register pair HL with the start of the string work area.
1B9EH - 1BA0H	Save the value in register pair HL as the next available location in the string work area pointer.
1BA1H - 1BA3H	Go set the current output device to the video display.
1BA4H - 1BA6H	Go turn off the cassette recorder.

1BA7H	Zero register A.
1BA8H	Zero register H.
1BA9H	Zero register L.
1BAAH - 1BACH	Save the value in register A as the current FOR flag.
1BADH	Save the value in register pair HL on the stack.
1BAEH	Save the value in register pair BC on the stack.
1BAFH - 1BB1H	Load register pair HL with the current value of the BASIC program pointer.
1BB2H	Return.
1BB3H - 1BBFH	**KEYBOARD INPUT ROUTINE**
1BB3H - 1BB4H	Load register A with a ?.
1BB5H - 1BB7H	Go display the ? in register A on the video display.
1BB8H - 1BB9H	Load register A with a space.
1BBAH - 1BBCH	Go display the space in register A on the video display.
1BBDH - 1BBFH	Jump to the keyboard input routine.
1BC0H - 1C8FH	**TOKENIZE INPUT ROUTINE**
1BC0H	Zero register A.
1BC1H - 1BC3H	Save the value in register A as the current value of the tokenization flag.
1BC4H	Zero register C.
1BC5H	Load register pair DE with the start of the input address in register pair HL.
1BC6H - 1BC8H	Load register pair HL with the starting address of the input buffer.
1BC9H	Decrement the value of the input buffer pointer in register pair HL.
1BCAH	Decrement the value of the input buffer pointer in register pair HL.
1BCBH	Exchange the value of the input buffer pointer in register pair HL with the value of the input pointer in register pair DE.
1BCCH	Load register A with the character at the location of the input pointer in register pair HL.

1BCDH - 1BCEH		Check to see if the character in register A is a space.
1BCFH - 1BD1H		Jump if the character in register A is a space.
1BD2H		Load register B with the value of the character in register A.
1BD3H - 1BD4H		Check to see if the character in register A is a quote.
1BD5H - 1BD7H		Jump if the character in register A is a quote.
1BD8H		Check to see if the character in register A is an end of the input character.
1BD9H - 1BDBH		Jump if the character in register A is an end of the input character.
1BDCH - 1BDEH		Load register A with the value of the tokenization flag.
1BDFH		Check to see if a DATA statement is being processed.
1BE0H		Load register A with the value of the character at the location of the input buffer pointer in register pair HL.
1BE1H - 1BE3H		Jump if a DATA statement is being processed.
1BE4H - 1BE5H		Check to see if the character in register A is a ?.
1BE6H - 1BE7H		Load register A with a PRINT token.
1BE8H - 1BEAH		Jump if the character at the location of the input buffer pointer in register pair HL is a ?.
1BEBH		Load register A with the character at the location of the input buffer pointer in register pair HL.
1BECH - 1BEDH		Check to see if the character in register A is less than a zero character.
1BEEH - 1BEFH		Jump if the character in register A is less than a zero character.
1BF0H - 1BF1H		Check to see if the character in register A is less than < character.
1BF2H - 1BF4H		Jump if the character in register A is less than < character.
1BF5H		Save the value of the input buffer pointer in register pair DE on the stack.
1BF6H - 1BF8H		Load register pair DE with the starting address of the reserved words list.

1BF9H	Save the value in register pair BC on the stack.
1BFAH - 1BFCH	Load register pair BC with the return address.
1BFDH	Save the return address in register pair BC on the stack.
1BFEH - 1BFFH	Load register B with the initial reserved words counter.
1B00H	Load register A with the character at the location of the input buffer pointer in register pair HL.
1C01H 1C01H	Check to see if the character in register A is lowercase.
1C03H - 1C04H	Jump if the character in register A isn't lowercase.
1C05H - 1C06H	Check to see if the character in register A is lowercase.
1C07H - 1C08H	Jump if the character in register A isn't lowercase.
1C09H - 1C0AH	Make the lowercase character in register A uppercase.
1C0BH	Save the adjusted character in register A at the location of the input buffer pointer in register pair HL.
1C0CH	Load register C with the character at the location of the input buffer pointer in register pair HL.
1C0DH	Exchange the input buffer pointer in register pair HL with the reserved words list pointer in register pair DE.
1C0EH	Bump the value of the reserved words list pointer in register pair HL.
1C0FH	Check to see if bit 7 of the character at the location of the reserved words list pointer in register pair HL is set.
1C10H - 1C12H	Jump if the character at the location of the reserved words list pointer in register pair HL doesn't have bit 7 set.
1C13H	Bump the value of the reserved words counter in register B.
1C14H	Load register A with the character at the location of the reserved words list pointer in register pair HL.
1C15H - 1C16H	Reset bit 7 of the character in register A.

1C17H	Return if this is the end of the reserved words list.
1C18H	Check to see if the character in register C is the same as the character at the location of the reserved words list pointer in register A.
1C19H - 1C1AH	Jump if the characters don't match.
1C1BH	Exchange the value of the reserved words list pointer in register pair HL with the value of the input buffer pointer in register pair DE.
1C1CH	Save the value of the input buffer pointer in register pair HL on the stack.
1C1DH	Bump the value of the reserved words list pointer in register pair DE.
1C1EH	Load register A with the character at the location of the reserved words list pointer in register pair DE.
1C1FH	Check to see if bit 7 of the character in register A is set.
1C20H - 1C22H	Jump if bit 7 of the character in register A is set.
1C23H	Load register C with the character in register A.
1C24H	Load register A with the value of the reserved words counter in register B.
1C25H - 1C26H	Check to see if the current reserved word being checked is GOTO.
1C27H - 1C28H	Jump if the current reserved word being checked isn't GOTO.
1C29H	Go bump the input buffer pointer in register pair HL till it points to the next character.
1C2AH	Decrement the value of the input buffer pointer in register pair HL.
1C2BH	Bump the value of the input buffer pointer in register pair HL.
1C2CH	Load register A with the character at the location of the input buffer pointer in register pair HL.
1C2DH - 1C2EH	Check to see if the character in register A is lowercase.
1C2FH - 1C30H	Jump if the character in register A isn't lowercase.
1C31H - 1C32H	Make the character in register A uppercase.

1C33H	Check to see if the character in register A matches the character in register C.
1C34H - 1C35H	Jump if the character in the input matches the reserved words character.
1C36H	Get the current input buffer pointer from the stack and put it in register pair HL.
1C37H - 1C38H	Jump.
1C39H	Load register C with the value of the reserved words counter in register B.
1C3AH	Get the value from the stack and put it in register pair AF.
1C3BH	Exchange the input buffer pointer in register pair HL with the reserved words list pointer in register pair DE.
1C3CH	Return.
1C3DH	Exchange the reserved words list pointer in register pair HL with the input buffer pointer in register pair DE.
1C3EH	Load register A with the value of the reserved words counter in register C.
1C3FH	Get the value from the stack and put it in register pair BC.
1C40H	Get the input buffer pointer from the stack and put it in register pair DE.
1C41H	Exchange the input buffer pointer in register pair HL with the input buffer pointer in register pair DE.
1C42H - 1C43H	Check to see if the token in register A is an ELSE token.
1C44H - 1C45H	Save a colon at the location of the input buffer pointer in register pair HL.
1C46H - 1C47H	Jump if the token in register A isn't an ELSE token.
1C48H	Bump the value of the counter in register C.
1C49H	Bump the value of the input buffer pointer in register pair HL.
1C4AH - 1C4BH	Check to see if the token in register A is a REM token.
1C4CH - 1C4DH	Jump if the token in register A isn't a REM token.

1C4EH - 1C4FH	Save a colon at the location of the input buffer pointer in register pair HL.
1C50H	Bump the value of the input buffer pointer in register pair HL.
1C51H - 1C52H	Load register B with a REM token.
1C53H	Save the REM token in register B at the location of the input buffer pointer in register pair HL.
1C54H	Bump the value of the input buffer pointer in register pair HL.
1C55H	Exchange the value of the input buffer pointer in register pair HL with the value of the input buffer pointer in register pair DE.
1C56H	Bump the value of the counter in register C.
1C57H	Bump the value of the counter in register C.
1C58H - 1C59H	Jump.
1C5AH	Exchange the value of the input buffer pointer in register pair HL with the value of the input buffer pointer in register pair DE.
1C5BH	Bump the value of the input buffer pointer in register pair HL.
1C5CH	Save the value of the token in register A at the location of the input buffer pointer in register pair DE.
1C5DH	Bump the value of the input buffer pointer in register pair DE.
1C5EH	Bump the value of the counter in register C.
1C5FH - 1C60H	Check to see if the character in register A is a colon.
1C61H - 1C62H	Jump if the character in register A is a colon.
1C63H - 1C64H	Check to see if the token in register A is a DATA token.
1C65H - 1C66H	Jump if the token in register A isn't a DATA token.
1C67H - 1C69H	Save the value in register A as the current tokenization flag.
1C6AH - 1C6BH	Check to see if the token in register A is a REM token.
1C6CH - 1C6EH	Jump if the token in register A isn't a REM token.

1C6FH	Load register B with a zero.
1C70H	Load register A with the character at the location of the input buffer pointer in register pair HL.
1C71H	Check to see if the character in register A is an end of the input character.
1C72H - 1C73H	Jump if the character in register A is an end of the input character.
1C74H	Check to see if the character in register B matches the character in register A.
1C75H - 1C76H	Jump if the character in register B matches the character in register A.
1C77H	Bump the value of the input buffer pointer in register pair HL.
1C78H	Save the character in register A at the location of the input buffer pointer in register pair DE.
1C79H	Bump the value of the counter in register C.
1C7AH	Bump the value of the input buffer pointer in register pair DE.
1C7BH - 1C7CH	Loop till a match is found.
1C7DH - 1C7FH	Load register pair HL with a five.
1C80H	Load register B with zero.
1C81H	Add the length of the input in register pair BC to the value in register pair HL.
1C82H	Load register B with the MSB of the value in register H.
1C83H	Load register C with the LSB of the value in register L.
1C84H - 1C86H	Load register pair HL with the address of the input buffer.
1C87H	Decrement the value of the input buffer pointer in register pair HL.
1C88H	Decrement the value of the input buffer pointer in register pair HL.
1C89H	Decrement the value of the input buffer pointer in register pair HL.
1C8AH	Zero the location of the input buffer pointer in register pair DE.

1C8BH	Bump the value of the input buffer pointer in register pair DE.
1C8CH	Zero the location of the input buffer pointer in register pair DE.
1C8DH	Bump the value of the input buffer pointer in register pair DE.
1C8EH	Zero the location of the input buffer pointer in register pair DE.
1C8FH	Return.
1C90H - 1C95H	**RST 0018H CODE**
1C90H	Load register A with the MSB of the value in register H.
1C91H	Subtract the value of the MSB of the value in register D from the MSB of the value in register A.
1C92H	Return if the MSB of the value in register D doesn't equal the MSB of the value in register H.
1C93H	Load register A with the LSB of the value in register L.
1C94H	Subtract the LSB of the value in register E from the LSB of the value in register A.
1C95H	Return.
1C96H - 1CA0H	**RST 0008H CODE**
1C96H	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
1C97H	Exchange the value of the current BASIC program pointer in register pair HL with the value of the return address on the stack.
1C98H	Check to see if the character at the location of the return address in register pair HL is the same as the character in register A.
1C99H	Bump the value of the return address in register pair HL.
1C9AH	Exchange the value of the return address in register pair HL with the value of the current BASIC program pointer on the stack.
1C9BH - 1C9DH	Jump to the RST 0010H code if the characters match.

1C93 - 1C90H	Jump to the Level II BASIC error routine and display a SN ERROR message if the characters don't match.
1CA1H - 1D1DH	**Level II BASIC FOR ROUTINE**
1CA1H - 1CA2H	Load register A with the value for the FOR flag.
1CA3H - 1CA5H	Save the value in register A as the current value of the FOR flag.
1CA6H - 1CA8H	Go evaluate the N=M portion of the expression.
1CA9H	Exchange the value of the current BASIC program pointer in register pair HL with the value on the stack.
1CAAH - 1CACH	Go check to see if there is a FOR statement on the stack already using the same variable name.
1CADH	Get the current BASIC program pointer from the stack and put it in register pair DE.
1CAEH - 1CAFH	Jump if there isn't a matching FOR statement on the stack.
1CB0H	Add the value in register pair BC to the value in register pair HL.
1CB1H	Load register pair SP with the value in register pair HL.
1CB2H - 1CB4H	Save the value in register pair HL as the stack pointer.
1CB5H	Exchange the value of the current BASIC program pointer in register pair DE with the value of the stack pointer address in register pair HL.
1CB6H - 1CB7H	Load register C with the number of words of memory to check for.
1CB8H - 1CBAH	Go check to see if there is enough memory left.
1CBBH	Save the value of the current BASIC program pointer in register pair HL on the stack.
1CBCH - 1CBEH	Go bump the current BASIC program pointer in register pair HL till it points to the end of the BASIC statement.
1CBFH	Exchange the adjusted value of the current BASIC program pointer in register pair HL with the value of the current BASIC program pointer on the stack.
1CC0H	Save the value of the current BASIC program pointer in register pair HL on the stack.

1CC1H - 1CC3H	Load register pair HL with the value of the current BASIC line number.
1CC4H	Exchange the value of the current BASIC line number in register pair HL with the value of the current BASIC program pointer on the stack.
1CC5H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be TO token(BD).
1CC6H	The value to be checked for by the syntax check is stored here.
1CC7H	Go check the current value of the number type flag.
1CC8H - 1CCAH	Go to the Level II BASIC error routine and display a TM ERROR message if the current value of the number type flag is a string.
1CCBH - 1CCDH	Go to the Level II BASIC error routine and display a TM ERROR message if the current value of the number type flag is double precision.
1CCEH	Save the value in register pair AF on the stack.
1CCFH - 1CD1H	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the result in REG1.
1CD2H	Get the value from the stack and put it in register pair AF.
1CD3H	Save the value of the current BASIC program pointer in register pair HL on the stack.
1CD4H - 1CD6H	Jump if the current number type is single precision.
1CD7H - 1CD9H	Go convert the current value in REG1 to an integer.
1CDAH	Exchange the integer value in register pair HL with the value of the current BASIC program pointer on the stack.
1CDBH - 1CCDH	Load register pair DE with the default STEP value.
1CDEH	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
1CDFH - 1CE0H	Check to see if the character in register A is a STEP token.
1CE1H - 1CE3H	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the integer value in register pair DE if the character in register A is a STEP token.

1CE4H	Save the integer value in register pair DE on the stack.
1CE5H	Save the current BASIC program pointer in register pair HL on the stack.
1CE6H	Exchange the integer value in register pair DE with the value of the current BASIC program pointer in register pair HL.
1CE7H - 1CE9H	Go load register A with the value of the sign for the STEP value in register pair HL.
1CEAH - 1CEBH	Jump.
1CECH - 1CEEH	Go convert the current value in REG1 to single precision.
1CEFH - 1CF1H	Go move the single precision value in REG1 into register pairs BC and DE.
1CF2H	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
1CF3H	Save the exponent and the MSB for the single precision value in register pair BC on the stack.
1CF4H	Save the NMSB and the LSB for the single precision value in register pair DE on the stack.
1CF5H - 1CF7H	Load register pair BC with the exponent and the MSB for a single precision constant.
1CF8H	Zero the NMSB for the single precision constant in register D.
1CF9H	Zero the LSB for the single precision constant in register E. Register pairs BC and DE now hold a single precision constant equal to one.
1CFAH	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
1CFBH - 1CFCH	Check to see if the character in register A is a STEP token.
1CFDH - 1CFEH	Load register A with the default sign for the STEP value.
1CFFH - 1D00H	Jump if the character at the location of the current BASIC program pointer in register A isn't a STEP token.
1D01H - 1D03H	Go evaluate the expression at the location of the current BASIC program pointer and return with the result in REG1.

1D04H		Save the value of the current BASIC program pointer in register pair HL on the stack.
1D05H	- 1D07H	Go convert the current value in REG1 to single precision and return with the result in REG1.
1D08H	- 1D0AH	Go move the single precision value in REG1 into register pairs BC and DE.
1D0BH	- 1D0DH	Go get the sign for the value in REG1 into register A.
1D0EH		Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
1D0FH		Save the exponent and the NMSB for the single precision value in register pair BC on the stack.
1D10H		Save the NMSB and the LSB for the single precision value in register pair DE on the stack.
1D11H		Load register C with the sign value in register A.
1D12H		Go check the current value of the number type flag.
1D13H		Load register B with the value of the number type flag test in register A.
1D14H		Save the value in register pair BC on the stack.
1D15H		Save the value of the current BASIC program pointer in register pair HL on the stack.
1D16H	- 1D18H	Load register B with a FOR token.
1D19H		Exchange the value of the variable address in register pair HL with the value of the current BASIC program pointer on the stack.
1D1AH	- 1D1BH	Load register B with the FOR token.
1D1CH		Save the value in register pair BC on the stack.
1D1DH		Bump the value of the stack pointer.
1D1EH	- 1D77H	**LEVEL II BASIC INTERPRETER**
1D1EH	- 1D20H	Go check to see if a key has been pressed.
1D21H		Set the flags.
1D22H	- 1D24H	Go check to see if the key pressed was a shift@.
1D25H	- 1D27H	Save the value of the current BASIC program pointer.
1D28H	- 1D2BH	Save the value of the stack pointer.

1D2CH	Load register A with the value at the location of the current BASIC program pointer in register pair HL.
1D2DH - 1D2EH	Check to see if the character in register A is a colon.
1D2FH - 1D30H	Jump if the character in register A is a colon.
1D31H	Check to see if the character in register A is an end of the BASIC line character.
1D32H - 1D34H	Go to the Level II BASIC error routine and display a SN ERROR message if the character in register A isn't an end of the BASIC line character.
1D35H	Bump the value of the current BASIC program pointer in register pair HL.
1D36H	Load register A with the LSB of the next BASIC line pointer at the location of the current BASIC program pointer in register pair HL.
1D37H	Bump the value of the current BASIC program pointer in register pair HL.
1D38H	Check to see if the next BASIC line pointer is equal to zero.
1D39H - 1D3BH	Jump if this is the end of the BASIC program.
1D3CH	Bump the value of the current BASIC program pointer in register pair HL.
1D3DH	Load register E with the LSB of the BASIC line number at the location of the current BASIC program pointer in register pair HL.
1D3EH	Bump the value of the current BASIC program pointer in register pair HL.
1D3FH	Load register D with the MSB of the BASIC line number at the location of the current BASIC program pointer in register pair HL.
1D40H	Exchange the value of the BASIC line number in register pair DE with the value of the current BASIC program pointer in register pair HL.
1D41H - 1D43H	Save the value of the BASIC line number in register pair HL.
1D44H - 1D46H	Load register A with the value of the TRON flag.
1D47H	Check for TRON.
1D48H - 1D49H	Jump if TROFF.

1D4AH	Save the value of the current BASIC program pointer in register pair DE on the stack.
1D4BH - 1D4CH	Load register A with a <.
1D4DH - 1D4FH	Go display the character in register A on the video display.
1D50H - 1D52H	Go display the current BASIC line number in register pair HL on the video display.
1D53H - 1D54H	Load register A with a >.
1D55H - 1D57H	Go display the character in register A on the video display.
1D58H	Get the value of the current BASIC program pointer from the stack and put it in register pair DE.
1D59H	Load register pair HL with the value of the current BASIC program pointer in register pair DE.
1D5AH	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
1D5BH - 1D5DH	Load register pair DE with the return address.
1D5EH	Save the return address in register pair DE on the stack.
1D5FH	Return if the character at the location of the current BASIC program pointer in register pair HL is an end of the BASIC line character.
1D60H - 1D61H	Check to see if the character at the location of the current BASIC program pointer in register A is a token.
1D62H - 1D64H	Jump if the character in register A isn't a BASIC token.
1D65H - 1D66H	Check to see if the token in register A is less than a TAB(token.
1D67H - 1D69H	Jump if the token in register A is greater than or equal to a TAB(token.
1D6AH	Multiply the token value in register A by two.
1D6BH	Load the adjusted token value in register C from register A.
1D6CH - 1D6DH	Load register B with zero.

1D6EH	Load register pair DE with the value of the current BASIC program pointer in register pair HL.
1D6FH - 1D71H	Load register pair HL with the list of BASIC execution addresses.
1D72H	Add the value of the token offset in register pair BC to the starting address of the list of BASIC execution addresses in register pair HL.
1D73H	Load register C with the LSB of the execution address at the location of the memory pointer in register pair HL.
1D74H	Bump the value of the memory pointer in register pair HL.
1D75H	Load register B with the MSB of the execution address at the location of the memory pointer in register pair HL.
1D76H	Save the value of the execution address in register pair BC on the stack.
1D77H	Load register pair HL with the value of the current BASIC program pointer in register pair DE.
1D78H - 1D90H	**RST 0010H CODE**
1D78H	Bump the value of the current BASIC program pointer in register pair HL.
1D79H	Load register A with the value of the character at the location of the current BASIC program pointer in register pair HL.
1D7AH - 1D7BH	Check to see if the character at the location of the current BASIC program pointer in register A is greater than or equal to a colon.
1D7CH	Return if the character at the location of the current BASIC program pointer in register A is greater than or equal to a colon.
1D7DH - 1D7EH	Check to see if the character at the location of the current BASIC program pointer in register A is a space.
1D7FH - 1D81H	Loop if the character at the location of the current BASIC program pointer in register A is a space.
1D82H - 1D83H	Check to see if the character at the location of the current BASIC program pointer in register A is greater than or equal to 0BH.
1D84H - 1D85H	Jump if the character at the location of the current

	BASIC program pointer in register A if greater than or equal to 0BH.
1D86H - 1D87H	Check to see if the character at the location of the current BASIC program pointer in register A is greater than or equal to 09H.
1D88H - 1D8AH	Loop if the character at the location of the current BASIC program pointer in register A is greater than or equal to 09H.
1D8BH - 1D8CH	Check to see if the character at the location of the current BASIC program pointer in register A is greater than or equal to a zero character.
1D8DH	Complement the value of the Carry flag.
1D8EH	Bump the value of the character at the location of the current BASIC program pointer in register A.
1D8FH	Decrement the value of the character at the location of the current BASIC program pointer in register A.
1D90H	Return.
1D91H - 1D9AH	**LEVEL II BASIC RESTORE ROUTINE**
1D91H	Load register pair DE with the value of the current BASIC program pointer in register pair HL.
1D92H - 1D94H	Load register pair HL with the start of the BASIC program pointer.
1D95H	Decrement the start of the BASIC program pointer in register pair HL.
1D96H - 1D98H	Save the adjusted value in register pair HL as the current DATA pointer.
1D99H	Load register pair HL with the value of the current BASIC program pointer in register pair DE.
1D9AH	Return.
1D9BH - 1DADH	**SCAN KEYBOARD ROUTINE**
1D9BH - 1D9DH	Go scan the keyboard.
1D9EH	Check to see if a key was pressed.
1D9FH	Return if a key wasn't pressed.
1DA0H - 1DA1H	Check to see if the key pressed in register A is a shift @ key.
1DA2H - 1DA4H	Go wait for another key to be pressed if the key pressed in register A was a shift @ key.

1DA5H - 1DA7H	Save the key pressed in register A as the value of the last key pressed.
1DA8H	Check to see if the BREAK key was pressed.
1DA9H	Return if the BREAK key wasn't pressed.
1DAAH	Readjust the value of the key pressed in register A.
1DABH - 1DADH	Jump.
1DAEH - 1DE3H	**LEVEL II BASIC END ROUTINE**
1DAEH	Return and display a SN ERROR message if there is anything following the END token.
1DAFH	Save the value in register pair AF on the stack.
1DB0H - 1DB2H	Go call the DOS link at 41BBH.
1DB3H	Get the value from the stack and put it in register pair AF.
1DB4H - 1DB6H	Save the value of the current BASIC program pointer in register pair HL.
1DB7H - 1DB9H	Load register pair HL with the starting address of the temporary string work area.
1DBAH - 1DBCH	Save the value in register pair HL as the new temporary string work area pointer.
1DBEH - 1DBFH	Set the flags.
1DC0H	Get the value from the stack and put it in register pair BC.
1DC1H - 1DC3H	Load register pair HL with the value of the current BASIC line number.
1DC4H	Save the value of the current BASIC line number in register pair HL on the stack.
1DC5H	Save the value in register pair AF on the stack.
1DC6H	Load register A with the LSB of the current BASIC line number in register L.
1DC7H	Combine the MSB of the current BASIC line number in register H with the LSB of the current BASIC line number in register A.
1DC8H	Bump the combined value of the current BASIC line number in register A.
1DC9H - 1DCAH	Jump if Level II BASIC is in the command mode.

1DCBH - 1DCDH	Save the value of the current BASIC line number in register pair HL.
1DCEH - 1DD0H	Load register pair HL with the value of the current BASIC program pointer.
1DD1H - 1DD3H	Save the value of the current BASIC program pointer in register pair HL.
1DD4H - 1DD6H	Go set the current output device to the video display.
1DD7H - 1DD9H	Go display a carriage return if necessary.
1DDAH	Get the value from the stack and put it in register pair AF.
1DDBH - 1DDDH	Load register pair HL with the starting address of the BREAK message.
1DDEH - 1DE0H	Jump if it was a BREAK or a STOP that halted program execution.
1DE1H - 1DE3H	Jump.
1DE4H - 1DF6H	**LEVEL II BASIC CONT ROUTINE**
1DE4H - 1DE6H	Load register pair HL with the value of the continuation address.
1DE7H	Load register A with the MSB of the continuation address in register H.
1DE8H	Combine the LSB of the continuation address in register L with the MSB of the continuation address in register A.
1DE9H - 1DEAH	Load register E with a CN ERROR code.
1DEBH - 1DEDH	Go to the Level II BASIC error routine and display a CN ERROR message if CONT isn't possible.
1DEEH	Load register pair DE with the value of the continuation address in register pair HL.
1DEFH - 1DF1H	Load register pair HL with the value of the last BASIC line number executed.
1DF2H - 1DF4H	Save the last line number executed in register pair HL as the current BASIC line number.
1DF5H	Load register pair HL with the value of the continuation address in register pair DE.
1DF6H	Return.
1DF7H - 1DF8H	**TRON ENTRY POINT**

1DF7H - 1DF8H	Load register A with a nonzero value.
1DF8H	**TROFF ENTRY POINT**
1DF8H	Zero register A.
1DF9H - 1DFCH	**COMMON CODE SHARED BY TRON AND TROFF**
1DF9H - 1DFBH	Save the value in register A as the current value of the TRON/TROFF flag.
1DFCH	Return.
1DFDH - 1DFFH	**DISK ROUTINE NOT USED BY LEVEL II BASIC.**
1E00H - 1E02H	**DEFSTR ENTRY POINT**
1E00H - 1E01H	Load register E with the string number type flag (03H).
1E03H - 1E05H	**DEFINT ENTRY POINT**
1E03H - 1E04H	Load register E with the integer number type flag (02H).
1E06H - 1E08H	**DEFSNG ENTRY POINT**
1E06H - 1E07H	Load register E with the single precision number type flag (04H).
1E09H - 1E0AH	**DEFDBL ENTRY POINT**
1E09H - 1E0AH	Load register E with the double precision number type flag (08H).
1E0BH - 1E3CH	**COMMON CODE SHARED BY DEFSTR, DEFINT, DEFSNG, AND DEFDBL**
1E0BH - 1E0DH	Go check to see if the character at the location of the current BASIC program pointer in register A is alphabetic.
1E0EH - 1E10H	Load register pair BC with a return address which will return to the SN ERROR routine.
1E11H	Save the value of the return address in register pair BC on the stack.
1E12H	Return if the character at the location of the current BASIC program pointer in register A isn't alphabetic.
1E13H - 1E14H	Subtract 41H from the letter's value in register A so that it will be in the range of 0-25.

1E15H	Load register C with the adjusted value in register A.
1E16H	Load register B with the adjusted value in register A.
1E17H	Go bump the value of the current BASIC program pointer in register pair HL until it points to the next character.
1E18H - 1E19H	Check to see if the character at the location of the current BASIC program pointer in register A is a -.
1E1AH - 1E1BH	Jump if the character at the location of the current BASIC program pointer in register A isn't a -.
1E1CH	Go bump the current BASIC program pointer in register pair HL until it points to the next character.
1E1DH - 1E1FH	Go check to see if the character in register A is alphabetic.
1E20H	Return if the character at the location of the current BASIC program pointer in register A isn't alphabetic.
1E21H - 1E22H	Subtract 41H from the letter's value in register A so that it will be in the range of 0-25.
1E23H	Load register B with the adjusted value in register A.
1E24H	Go bump the value of the current BASIC program pointer in register pair HL until it points to the next character.
1E25H	Load register A with the value of the second letter in register B.
1E26H	Subtract the value of the first letter in register C from the value of the second letter in register A.
1E27H	Return if the letter values are in descending order.
1E28H	Bump the value in register A so that it holds the number of variable names to be changed.
1E29H	Exchange the value of the return address on the stack with the value of the current BASIC program pointer in register pair HL.
1E2AH - 1E2CH	Load register pair HL with the starting address of the variable declaration table.
1E2DH - 1E2EH	Load register B with zero.
1E2FH	Add the value of the first letter in register pair BC to

	the value of the starting address of the variable declaration table in register pair HL.
1E30H	Save the number type flag in register E at the location of the memory pointer in register pair HL.
1E31H	Bump the value of the memory pointer in register pair HL.
1E32H	Decrement the count of the number of variables to be changed in register A.
1E33H - 1E34H	Loop until all of the variables have been changed.
1E35H	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
1E36H	Load A with the character at the location of the current BASIC program pointer in register pair HL.
1E37H - 1E38H	Check to see if the character at the location of the current BASIC program pointer in register A is a comma.
1E39H	Return if the character at the location of the current BASIC program pointer in register A isn't a comma.
1E3AH	Go bump the current BASIC program pointer in register pair HL until it points to the next character.
1E3BH - 1E3CH	Loop until done.
1E3DH - 1E44H	**EXAMINE VARIABLE**
1E3DH	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
1E3EH - 1E3FH	Check to see if the character at the location of the current BASIC program pointer in register A is less than an A.
1E40H	Return if the character at the location of the current BASIC program pointer in register A is less than an A.
1E41H - 1E42H	Check to see if the character at the location of the current BASIC program pointer in register A is greater than a Z.
1E43H	Complement the value of the Carry flag. On exit this routine will have the Carry flag set if the character at the location of the current BASIC program pointer in register A isn't alphabetic and will have the Carry flag cleared if the character at the location of the current BASIC program pointer in register A is alphabetic.

1E44H	Return.
1E45H - 1E4EH	**EXAMINE VARIABLE**
1E45H	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
1E46H - 1E48H	Go evaluate the expression at the current location of the BASIC program pointer in register pair HL and return with the integer result in register pair DE.
1E49H	Return if the integer result in register pair DE is positive.
1E4AH - 1E4BH	Load register E with an FC ERROR code.
1E4CH - 1E4EH	Go to the Level II BASIC error routine and display a FC ERROR message if the integer result in register pair DE is negative.
1E4FH - 1E79H	**ASCII TO BINARY**
1E4FH	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
1E50H - 1E51H	Check to see if the character at the location of the current BASIC program pointer in register A is a period.
1E52H	Load register pair DE with the value of the current BASIC program pointer in register pair HL.
1E53H - 1E55H	Load register pair HL with the current BASIC line number.
1E56H	Exchange the value of the current BASIC program pointer in register pair DE with the current BASIC line number in register pair HL.
1E57H - 1E59H	Jump if the character at the location of the current BASIC program pointer in register A is a period.
1E5AH	Decrement the value of the current BASIC program pointer in register pair HL.
1E5BH - 1E5DH	Load register pair DE with zero.
1E5EH	Go bump the value of the current BASIC program pointer in register pair HL until it points to the next character.
1E5FH	Return if the character at the location of the current BASIC program pointer in register A isn't numeric.
1E60H	Save the value of the current BASIC program pointer in register pair HL on the stack.

1E61H	Save the value in register pair AF on the stack.
1E62H - 1E64H	Load register pair HL with 6552.
1E65H	Check to see if the integer value in register pair DE is greater than 6552.
1E66H - 1E68H	Go to the Level II BASIC error routine and display a SN ERROR message if the value in register pair DE is greater than 6552.
1E69H	Load register H with the MSB of the integer total in register D.
1E6AH	Load register L with the LSB of the integer total in register E.
1E6BH	Multiply the integer value in register pair HL by two.
1E6CH	Multiply the integer value in register pair HL by two. The integer result in register pair HL is now equal to the integer value in register pair DE times four.
1E6DH	Add the integer value in register pair DE to the integer value in register pair HL. The integer result in register pair HL is now equal to the integer value in register pair DE times five.
1E6EH	Multiply the integer value in register pair HL by two. The integer result in register pair HL is now equal to the integer value in register pair DE times ten.
1E6FH	Get the value from the stack and put it in register pair AF.
1E70H - 1E71H	Convert the ASCII digit in register A to binary.
1E72H	Load register E with the binary value of the character in register A.
1E73H - 1E74H	Load register D with zero.
1E75H	Add the binary value of the character in register pair DE to the integer value in register pair HL.
1E76H	Load register pair DE with the integer result in register pair HL.
1E77H	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
1E78H - 1E79H	Loop till the ASCII to binary conversion has been completed.

1E7AH - 1EA0H	**LEVEL II BASIC CLEAR ROUTINE**
1E7AH - 1E7CH	Jump if there isn't a number of bytes specified to clear for string space.
1E7DH - 1D7FH	Go evaluate the number of bytes to be reserved for string space and return with the integer result in register pair DE.
1D80H	Decrement the current BASIC program pointer in register pair HL.
1E81H	Go bump the value of the current BASIC program pointer in register pair HL until it points to the next character.
1E82H	Return if the character at the location of the current BASIC program pointer isn't an end of the BASIC statement character.
1E83H	Save the value of the current BASIC program pointer in register pair HL on the stack.
1E84H - 1E86H	Load register pair HL with the top of memory pointer.
1E87H	Load register A with the LSB of the top of memory pointer in register L.
1E88H	Subtract the LSB of the number of bytes for string space in register E from the LSB of the top of memory pointer in register A.
1E89H	Load register E with the LSB of the start of string space in register A.
1E8AH	Load register A with the MSB of the top of memory pointer in register H.
1E8BH	Subtract the MSB of the number of bytes for string space in register D from the MSB of the top of memory pointer in register A.
1E8CH	Load register D with the MSB of the start of string space pointer in register A.
1E8DH - 1E8FH	Go to the Level II BASIC error routine and display an OM ERROR message if there isn't enough memory for the amount of string space specified.
1E90H - 1E92H	Load register pair HL with the end of the BASIC program pointer.
1E93H - 1E95H	Load register pair BC with the least amount of space needed for BASIC program variables.
1E96H	Add the value in register pair BC to the end of

	BASIC program pointer in register pair HL.
1E97H	Go compare the adjusted end of the BASIC program pointer in register pair HL to the start of string space pointer in register pair DE.
1E98H - 1E9AH	Go to the Level II BASIC error routine and display an OM ERROR message if the start of string space pointer in register pair DE is less than the adjusted end of the BASIC program pointer in register pair HL.
1E9BH	Load register pair HL with the start of string space pointer in register pair DE.
1E9CH - 1E9EH	Save the start of string space pointer in register pair HL.
1E9FH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
1EA0H - 1EA2H	Jump.
1EA3H - 1EB0H	**LEVEL II BASIC RUN ROUTINE**
1EA3H - 1EA5H	Jump if there isn't a line number specified after the RUN token.
1EA6H - 1EA8H	Call the DOS line at 41C7H.
1EA9H - 1EABH	Go initialize the BASIC variables and pointers.
1EACH - 1EAEH	Load register pair BC with the return address.
1EAFH - 1EB0H	Jump.
1EB1H - 1EC1H	**LEVEL II BASIC GOSUB ROUTINE**
1EB1H - 1EB2H	Load register B with the number of bytes needed for the GOSUB push.
1EB3H - 1EB5H	Go make sure that there's enough memory for the GOSUB push.
1EB6H	Get the return address from the stack and put it in register pair BC.
1EB7H	Save the value of the current BASIC program pointer in register pair HL on the stack.
1EB8H	Save the value of the current BASIC program pointer on the stack.
1EB9H - 1EBBH	Load register pair HL with the value of the current BASIC line number.

1EBCH	Exchange the value of the current BASIC program pointer on the stack with the value of the current BASIC line number in register pair HL.
1EBDH - 1EBEH	Load register A with a GOSUB token.
1EBFH	Save the value in register pair AF on the stack.
1EC0H	Bump the value of the stack pointer.
1EC1H	Save the return address in register pair BC on the stack.
1EC2H - 1EDDH	**LEVEL II BASIC GOTO ROUTINE**
1EC2H - 1EC4H	Go evaluate the line number at the location of the current BASIC program pointer in register pair HL and return with the result in register pair DE.
1EC5H - 1EC7H	Go bump the current BASIC program pointer in register pair HL till it points to the end of the current BASIC line.
1EC8H	Save the value of the current BASIC program pointer in register pair HL on the stack.
1EC9H - 1ECBH	Load register pair HL with the value of the current BASIC line number.
1ECCH	Go compare the value of the current BASIC line number in register pair HL with the value of the line number to branch to in register pair DE.
1ECDH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
1ECEH	Bump the value of the current BASIC program pointer in register pair HL.
1ECFH - 1ED1	If the line number to branch to in register pair DE is greater than the current BASIC line number then go find where the line number is located.
1ED2H - 1ED4H	If the line number to branch to in register pair DE is less than the current BASIC line number, then go find where the line number is located.
1ED5H	Load register H with the MSB of the new BASIC program pointer in register B.
1ED6H	Load register L with the LSB of the new BASIC program pointer in register C.
1ED7H	Decrement the value of the current BASIC program pointer in register pair HL.

1ED8H Jump if the new line number was found.

1ED9H - 1EDAH Load register E with an UL ERROR code.

1EDBH - 1EDDH Go to the Level II BASIC error routine and display an UL ERROR message if the line number wasn't found.

1EDEH - 1E04H **LEVEL II BASIC RETURN ROUTINE**

1EDEH Go to the Level II BASIC error routine and display a SN ERROR message if there is anything following the RETURN token.

1EDFH - 1EE0H Load register D with FFH

1EE1H - 1EE3H Go backspace the stack pointer by four and return with the value at the location of the stack pointer in register A.

1EE4H Load the stack pointer with the new value in register pair HL.

1EE5H - 1EE7HS Save the stack pointer position in register pair HL.

1EE8H - 1EE9H Check to see if the value in register A is a GOSUB token.

1EEAH - 1EEBH Load register E with a RG ERROR code.

1EECH - 1EEEH Go to the Level II BASIC error routine and display a RG ERROR message if there isn't a GOSUB push on the stack.

1EEFH Get the value of the GOBUS line number from the stack and put it in register pair HL.

1EF0H - 1EF2H Save the GOSUB line number in register pair HL as the current BASIC line number.

1EF3H Bump the value of the current BASIC line number in register pair HL.

1EF4H Load register A with the MSB of the adjusted current BASIC line number in register pair HL.

1EF5H Combine the LSB of the adjusted current BASIC line number in register L with the adjusted MSB of the current BASIC line number in register A.

1EF6H - 1EF7H Jump if Level II BASIC is in the command mode.

1EF8H - 1EFAH Load register A with the command mode flag.

1EFBH Check for the command mode.

1EFCH - 1EFEH	Jump if Level II BASIC is in the command mode.
1EFFH - 1F01H	Load register pair HL with the return address.
1F02H	Exchange the value of the current BASIC program pointer on the stack with the return address in register pair HL.
1F05H - 1F20H	**SCAN ROUTINE**
1F07H - 1F08H	Load register C with zero.
1F09H - 1F0AH	Load register B with zero.
1F0BH	Load register A with the stop scan character in register C.
1F0CH	Load register C with the stop scan character in register B.
1F0DH	Load register B with the stop scan character in register A.
1F0EH	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
1F0FH	Check to see if the character at the location of the current BASIC program pointer in register A is an end of the BASIC line character.
1F10H	Return if the character at the location of the current BASIC program pointer in register A is an end of the BASIC line character.
1F11H	Check to see if the character at the location of the current BASIC program pointer in register A is the same as the stop scan character in register B.
1F12H	Return if the character at the location of the current BASIC program pointer in register A is the same as the stop scan character in register B.
1F13H	Bump the value of the current BASIC program pointer in register pair HL.
1F14H - 1F15H	Check to see if the character at the location of the current BASIC program pointer in register A is a quote.
1F16H - 1F17H	Loop if the character at the location of the current BASIC program pointer in register A is a quote.
1F18H - 1F19H	Check to see if the character at the location of the current BASIC program pointer in register A is an IF token.

1F1AH - 1F1BH	Loop if the character at the location of the current BASIC program pointer in register A isn't an IF token.
1F1CH	Check to see if the character at the location of the current BASIC program pointer in register A is the same as the character in register B.
1F1DH	Add the value in register D to the value in register A.
1F1EH	Load register D with the value in register A.
1F1FH - 1F20H	Loop until scan is completed.
1F21H - 1F6BH	**LEVEL II BASIC LET ROUTINE**
1F21H - 1F23H	Go examine the variable at the location of the current BASIC program pointer in register pair HL and return with the variable's address in register pair DE.
1F24H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be an equal token (D5).
1F25H	The character to be checked for is stored here.
1F26H	Exchange the address of the variable in register pair DE with the value of the current BASIC program pointer in register pair HL.
1F27H - 1F29H	Save the address of the variable in register pair HL.
1F2AH	Exchange the value of the current BASIC program pointer in register pair DE with the address of the variable in register pair HL.
1F2BH	Save the address of the variable in register pair DE on the stack.
1F2CH	Go check the current value of the number type flag.
1F2DH	Save the value in register pair AF on the stack.
1F2EH - 1F30H	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the result in REG1.
1F31H	Get the value from the stack and put it in register pair AF.
1F32H	Exchange the address of the variable on the stack with the value of the current BASIC program pointer in register pair HL.
1F33H - 1F34H	Adjust the value in register A so that it will hold the correct number type flag.

1F35H - 1F37H	Go convert the result in REG1 to the same number type for the variable.
1F38H - 1F3AH	Go save the result of the expression.
1F3BH	Save the address of the variable in register pair DE on the stack.
1F3CH - 1F3DH	Jump if the result of the expression wasn't a string.
1F3EH - 1F40H	Load register pair HL with the starting address of the string's VARPTR in REG1.
1F41H	Save the VARPTR for the string result in register pair HL on the stack.
1F42H	Bump the value of the VARPTR for the string result in register pair HL.
1F43H	Load register E with the LSB of the address for the string at the location of the VARPTR for the string in register pair HL.
1F44H	Bump the value of the VARPTR for the string result in register pair HL.
1F45H	Load register D with the MSB of the address for the string at the location of the VARPTR for the string in register pair HL.
1F46H - 1F48H	Load register pair HL with the start of the BASIC program.
1F49H	Go compare the start of the BASIC program area in register pair HL with the address of the string result in register pair DE.
1F4AH - 1F4BH	Jump if the address of the string result in register pair DE is less than the start of the BASIC program area in register pair HL.
1F4CH - 1F4EH	Load register pair HL with the start of the string space pointer.
1F4FH	Go compare the start of the string spacer pointer in register pair HL with the address of the string result in register pair DE.
1F50H	Get the VARPTR for the string result from the stack and put it in register pair DE.
1F51H - 1F52H	Jump if the address of the string result in register pair DE was less than the start of the string space pointer in register pair HL.
1F53H - 1F55H	Load register pair HL with the simple variables pointer.

1F56H	Go compare the address of the VARPTR for the string result in register pair DE with the simple variables pointer in register pair HL.
1F57H - 1F58H	Jump if the address of the VARPTR for the string result in register pair DE is less than the simple variables pointer in register pair HL.
1F5AH	Get the VARPTR for the string result from the stack and put it in register pair DE.
1F5BH - 1F5DH	Go adjust the temporary string work area pointers.
1F5EH	Exchange the VARPTR for the string in register pair DE with the string's address in register pair HL.
1F5FH - 1F61H	Go move the string into string space.
1F62H - 1F64H	Go adjust the temporary string work area pointers.
1F65H	Exchange the address of the variable in register pair HL with the address of the variable on the stack.
1F66 - 1F68H	Go move the result to it's proper location in memory.
1F69H	Get the address of the variable from the stack and put it in register pair DE.
1F6AH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
1F6BH	Return.
1F6CH - 1FAEH	**LEVEL II BASIC ON ROUTINE**
1F6CH - 1F6DH	Check to see if the character at the location of the current BASIC program pointer in register A is an ERROR token.
1F6EH - 1F6FH	Jump if the character at the location of the current BASIC program pointer in register A isn't an ERROR token.
1F70H	Bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
1F71H - 1F72H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a GOTO token.
1F73H - 1F75H	Go evaluate the line number to branch to at the location of the current BASIC program pointer in register pair HL and return with the result in register pair DE.

1F76H	Load register A with the MSB of the line number in register D.
1F77H	Combine the LSB of the line number in register E with the MSB of the line number in register A.
1F78H - 1F79H	Jump if the line number in register pair DE is equal to zero.
1F7AH - 1F7CH	Go find the location of the line number in register pair DE and return with the location of the line number in register pair BC.
1F7DH	Load register D with the MSB of the location of the BASIC line in register B.
1F7EH	Load register E with the LSB of the location of the BASIC line in register C.
1F7FH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
1F80H - 1F82H	Go to the Level II BASIC error routine and display an UL ERROR message if the BASIC line doesn't exist.
1F83H	Exchange the location of the BASIC line in register pair DE with the value of the current BASIC program pointer in register pair HL.
1F84H - 1F86H	Save the location of the BASIC line in register pair HL.
1F87H	Exchange the value of the current BASIC program pointer in register pair DE with the location of the BASIC line in register pair HL.
1F88H	Return.
1F89H - 1F8BH	Load register A with the value of the error flag.
1F8CH	Check to see if the error flag is set.
1F8DH	Return if the error flag is set.
1F8EH - 1F90H	Load register A with the error code.
1F91H	Load register E with the value of the error code in register A.
1F92H - 1F94H	Go to the Level II error routine and display the error message for the error code in register E.
1F95H - 1F97H	Go evaluate the expression at the location of the current BASIC program pointer and return with the result in register pair DE.

1F98H		Load register A with the character at the location of the current BASIC program pointer in register pair HL.
1F99H		Load register B with the character at the location of the current BASIC program pointer in register A.
1F9AH	- 1F9BH	Check to see if the character at the location of the current BASIC program pointer in register A is a GOSUB token.
1F9CH	- 1F9DH	Jump if the character at the location of current BASIC program pointer in register A is a GOSUB token.
1F9EH	- 1F9FH	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a GOTO token.
1FA0H		Decrement the value of the current BASIC program pointer in register pair HL.
1FA1H		Load register C with the LSB of the expression after the ON token in register E.
1FA2H		Decrement the line number counter in register C.
1FA3H		Load register A with the token in register B.
1FA4H	- 1FA6H	Jump if the line number has been found.
1FA7H	- 1FA9H	Evaluate the line number at the location of the current BASIC program pointer and return with the result in register pair DE.
1FAAH	- 1FABH	Check to see if the character at the location of the current BASIC program pointer in register A is a comma.
1FACH		Return if the character at the location of the current BASIC program pointer in register A isn't a comma.
1FADH	- 1FAEH	Loop until the line number has been located.
1FAFH	- 1FF3H	**LEVEL II BASIC RESUME ROUTINE**
1FAFH	- 1FB1H	Load register pair DE with the address of the Level II BASIC error flag.
1FB2H		Load register A with the error flag at the location of the memory pointer in register pair DE.
1FB3H		Check for an error.
1FB4H	- 1FB6H	Go to the Level II BASIC error routine and display an RW ERROR message if an error hasn't occurred.

1FB7H	Clear the error flag in register A.
1FB8H - 1FBAH	Save the new error flag in register A.
1FBBH	Save the new error flag in register A at the location of the memory pointer in register pair DE.
1FBCH	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
1FBDH - 1FBEH	Check to see if the character at the location of the current BASIC program pointer in register A is a NEXT token.
1FBFH - 1FC0H	Jump if the character at the location of the current BASIC program pointer in register A is a NEXT token.
1FC1H - 1FC3H	Go evaluate the line number at the location of the current BASIC program pointer and return with the result in register pair DE.
1FC4H	Return if there wasn't a line number at the location of the current BASIC program pointer.
1FC5H	Load register A with the MSB of the line number in register D.
1FC6H	Combine the LSB of the line number in register E with the MSB of the line number in register A.
1FC7H - 1FC9H	Jump if the line number in register pair DE isn't equal to zero.
1FCAH	Bump the value in register A so that it will indicate RESUME0.
1FCBH - 1FCCH	Jump to the RESUME0 code.
1FCDH	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
1FCEH	Return if this is the end of the BASIC statement.
1FCFH - 1FD1H	Get the value of the current BASIC program pointer and put it in register pair HL.
1FD2H	Load register pair DE with the value of the current BASIC program pointer in register pair HL.
1FD3H - 1FD5H	Load register pair HL with the current BASIC line number.
1FD6H - 1FD8H	Save the value of the current BASIC line number in register pair HL.

1FD9H	Load register pair HL with the value of the current BASIC program pointer in register pair DE.
1FDAH	Return if this is RESUME0.
1FDBH	Get the character at the location of the current BASIC program pointer in register pair HL and put it in register A.
1FDCH	Check the character at the location of the current BASIC program pointer in register A to see if it's an end of the BASIC line character.
1FDDH - 1FDEH	Jump if the character at the location of the current BASIC program pointer in register A isn't an end of the BASIC line character.
1FDFH	Bump the value of the current BASIC program pointer in register pair HL.
1FE0H	Bump the value of the current BASIC program pointer in register pair HL.
1FE1H	Bump the value of the current BASIC program pointer in register pair HL.
1FE2H	Bump the value of the current BASIC program pointer in register HL.
1FE3H	Bump the value of the current BASIC program pointer in register pair HL.
1FE4H	Load register A with the MSB of the line number with the error in register D.
1FE5H	Combine the LSB of the line number with the error in register E with the MSB of the line number with the error in register A.
1FE6H	Bump the combined value of the line number with the error in register A.
1FE7H - 1FE9H	Jump if Level II BASIC isn't in the command mode.
1FEAH - 1FECH	Load register A with the command mode flag.
1FEDH	Check to see if the command mode flag in register A is set.
1FEEH - 1FF0H	Jump if Level II Basic is in the command mode.
1FF1H - 1FF3H	Return.
1FF4H - 2007H	**LEVEL II BASIC ERROR ROUTINE**
1FF4H - 1FF6H	Go evaluate the expression at the location of the

	current BASIC program pointer in register pair HL and return with the result in A.
1FF7H	Return if this isn't the end of the BASIC statement.
1FF8H	Check to see if the error number in register A is equal to zero.
1FF9H - 1FFBH	Go to the Level II BASIC error routine and display an FC ERROR message if the error code in register A is equal to zero.
1FFCH	Subtract one from the error code in register A.
1FFDH	Multiply the error code in register A by two.
1FFEH	Load register E with the value of the error code in register A.
1FFFH - 2000H	Check to see if the error code in register A is less than 45.
2001H - 2002H	Jump if the error code in register A is less than 45.
2003H - 2004H	Load register E with an UE ERROR code.
2005H - 2007H	Go to the Level II BASIC error routine and display the appropriate error message.
2008H - 2038H	**LEVEL II BASIC AUTO ROUTINE**
2008H - 200AH	Load register pair DE with a default line number of ten.
200BH	Save the default line number in register pair DE on the stack.
200CH - 200DH	Jump if this is the end of the BASIC statement.
200EH - 2010H	Go evaluate the line number at the current location of the BASIC program pointer in register pair HL and return with the result in register pair DE.
2011H	Exchange the value of the line number in register pair DE with the value of the current BASIC program pointer in register pair HL.
2012H	Exchange the default line number on the stack with the line number in register pair HL.
2013H - 2014H	Jump if this is the end of the BASIC statement.
2015H	Load register pair HL with the value of the current BASIC program pointer in register pair DE.
2016H - 2017H	Go check the syntax. The character at the location of

the current BASIC program pointer in register pair HL must be a comma.

2018H Load register pair DE with the value of the current BASIC program pointer in register pair HL.

2019H - 201BH Load register pair HL with the last AUTO increment value.

201CH Exchange the value of the current BASIC program pointer in register pair DE with the last AUTO increment value in register pair HL.

201DH - 201EH Jump if this is the end of the BASIC statement.

201FH - 2021H Go evaluate the AUTO increment number at the location of the current BASIC program pointer in register pair HL and return with the result in register pair DE.

2022H - 2024H Go to the Level II BASIC error routine and display a SN ERROR message if this isn't the end of the BASIC statement.

2025H Exchange the AUTO increment number in register pair DE with the value of the current BASIC program pointer in register pair HL.

2026H Load register A with the MSB of the AUTO increment number in register H.

2027H Combine the LSB of the AUTO increment number in register L with the MSB of the AUTO increment number in register A.

2028H - 202AH Go to the Level II BASIC error routine and display a FC ERROR message if the AUTO increment number in register pair HL is equal to zero.

202BH - 202DH Save the value of the AUTO increment number in register pair HL.

202EH - 2030H Set the AUTO flag.

2031H Get the line number from the stack and put it in register pair HL.

2032H - 2034H Save the line number in register pair HL as the next AUTO line number.

2035H Clean up the stack.

2036H - 2038H Jump to the Level II BASIC command mode.

2039H - 2066H **LEVEL II BASIC IF ROUTINE**

2039H - 203BH Go evaluate the expression at the location of the

	current BASIC program pointer and return with the result in REG1.
203CH	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
203DH - 203EH	Check to see if the character at the location of the current BASIC program pointer in register A is a comma.
203FH - 2041H	If the character at the location of the current BASIC program pointer in register A is a comma then go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
2042H - 2043H	Check to see if the character at the location of the current BASIC program pointer in register A is a THEN token.
2044H - 2046H	If the character at the location of the current BASIC program pointer in register A is a THEN token, then go bump the value of the current BASIC program pointer till it points to the next character.
2047H	Decrement the value of the current BASIC program pointer in register pair HL.
2048H	Save the value of the current BASIC program pointer on the stack.
2049H - 204BH	Check to see if the expression after the IF token was true or false.
204CH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
204DH - 204EH	Jump if the expression was false.
204FH	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
2050H - 2052H	Jump to the GOTO routine if the character at the location of the current BASIC program pointer in register pair HL is numeric.
2053H - 2055H	Jump.
2056H - 2057H	Load register D with the scan counter.
2058H - 205AH	Go scan the BASIC statement.
205BH	Check to see if this is the end of the BASIC instruction.

205CH	Return if this is the end of the BASIC statement.
205DH	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
205EH - 205FH	Check to see if the character at the location of the current BASIC program pointer in register A is an ELSE token.
2060H - 2061H	Loop till an ELSE token is found.
2062H	Decrement the value of the scan counter.
2063H - 2064H	Loop till all of the ELSE tokens have been found.
2065H - 2066H	Jump.
2067H - 206EH	**LEVEL II BASIC LPRINT ROUTINE**
2067H - 2068H	Load register A with the output device code for the printer.
2069H - 206BH	Save the value in register A as the current output device type number.
206CH - 206EH	Jump.
206FH - 2177H	**LEVEL II BASIC PRINT ROUTINE**
206FH - 2071H	Call the DOS link at 41CAH.
2072H - 2073H	Check the character at the location of the current BASIC program pointer in register A to see if it's an @.
2074H - 2075H	Jump if the character at the location of the current BASIC program pointer in register A isn't an @.
2076H - 2078H	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the result in register pair DE.
2079H - 207AH	Check to see if the location in register pair DE is greater than 1023H.
207AH - 207DH	Go to the Level II BASIC error routine and display a FC ERROR message if the location in register pair DE is greater than 1023.
207EH	Save the value of the current BASIC program pointer in register pair HL on the stack.
207FH - 2081H	Load register pair HL with the starting address of video memory.

2082H	Add the location in register pair DE to the starting address of video memory in register pair HL.
2083H - 2085H	Save the adjusted value in register pair HL as the current cursor position.
2086H	Load register A with the LSB of the new cursor position in register L.
2087H - 2088H	Mask the LSB of the cursor position in register A to determine the cursor's line position.
2089H - 208BH	Save the new cursor line position in register A.
208CH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
208DH - 208EH	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a comma.
208FH - 2090H	Check to see if the character at the location of the current BASIC program pointer in register A is a #.
2091H - 2092H	Jump if the character at the location of the current BASIC pointer in register A isn't a #.
2093H - 2095H	Go set the specified cassette drive number as the current cassette drive.
2096H - 2097H	Load register A with the output device type code for the cassette recorder.
2098H - 209AH	Save the value in register A as the current output device type code.
209BH	Decrement the value of the current BASIC program pointer in register pair HL.
209CH	Go bump the value of the current BASIC program pointer till it points to the next character.
209DH - 209FH	If the character at the location of the current BASIC program pointer in register A is an end of the BASIC statement character then go print a carriage return on the video display if necessary.
20A0H - 20A2H	Jump if the character at the location of the current BASIC program pointer in register A is an end of the BASIC statement character.
20A3H - 20A4H	Check to see if the character at the location of the current BASIC program pointer in register A is a USING token.

20A5H - 20A7H Jump if the character at the location of the current BASIC program pointer in register A is a USING token.

20A8H - 20A9H Check to see if the character at the location of the current BASIC program pointer in register A is a TAB(token.

20AAH - 20ACH Jump if the character at the location of the current BASIC program pointer in register A is a TAB(token.

20ADH Save the value of the current BASIC program pointer in register pair HL on the stack.

20AEH - 20AFH Check to see if the character at the location of the current BASIC program pointer in register A is a comma.

20B0H - 20B2H Jump if the character at the location of the current BASIC program pointer in register A is a comma.

20B3H - 20B4H Check to see if the character at the location of the current BASIC program pointer in register A is a semicolon.

20B5H - 20B7H Jump if the character at the location of the current BASIC program pointer in register A is a semicolon.

20B8H Get the value of the current BASIC program pointer from the stack and put it in register pair BC.

20B9H - 20BBH Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the result in REG1.

20BCH Save the value of the current BASIC program pointer in register pair HL on the stack.

20BDH Go check the current value of the number type flag.

20BEH - 20BFH Jump if the result in REG1 is a string.

20C0H - 20C2H Go convert the numeric value in REG1 to an ASCII string.

20C3H - 20C5H Go set up pointers in the temporary string area.

20C6H - 20C8H Call the DOS link at location 41CDH.

20C9H - 20CBH Load register pair HL with the VARPTR for the string to be sent to the current output device.

20CCH - 20CEH Load register A with the current value of the output device flag.

20CFH	Test the value of the current output device code in register A.
20D0H - 20D2H	Jump if the current output device is the cassette recorder.
20D3H - 20D4H	Jump if the current output device is the video display.
20D5H - 20D7H	Load register A with the current carriage position.
20D8H	Add the length of the string to be sent to the printer at the location of the memory pointer in register pair HL to the current carriage position in register A.
20D9H - 20DAH	Check to see if the adjusted length in register A is greater than 132.
20DBH - 20DCH	Jump.
20DDH - 20DFH	Load register A with the video line size.
20E0H	Load register B with the video line size in register A.
20E1H - 20E3H	Load register A with the current video line position.
20E4H	Add the length of the string to be sent to the video display at the location of the memory pointer in register pair HL to the value of the current video line position in register A.
20E5H	Check to see if the length in register A is greater than the video line length.
20E6H - 20E8H	Go send a carriage return to the current output device if necessary.
20E9H - 20EBH	Go send the string to the current output device.
20ECH - 20EDH	Load register A with a space.
20EEH - 20F0H	Go send the space in register A to the current output device.
20F1H	Check to see if register A is equal to zero.
20F2H - 20F4H	If necessary go send the string to the current output device.
20F6H - 20F8H	Loop till done.
20F9H - 20FBH	Load register A with the number of characters printed on the current line.
20FCH	Check to see if any characters have been printed on the current line.

20FDH	Return if no characters have been printed on the current line.
20FEH - 20FFH	Load register A with a carriage return.
21C0H - 2102H	Go send the carriage return in register A to the current output device.
2103H - 2105H	Call the DOS link at 41D0H.
2106H	Zero register A.
2107H	Return.
2108H - 210AH	Call the DOS link at 41D3H.
210BH - 210DH	Load register A with the value of the current output device flag.
210EH	Test the value of the current output device flag in register A.
210FH - 2111H	Jump if the printer or the video display is the current output device.
2112H - 2113H	Load register A with a comma.
2114H - 2116H	Send the comma in register A to the cassette recorder.
2117H - 2118H	Loop till done.
2119H - 211AH	Jump if the video display is the current output device.
211BH - 211DH	Load register A with the current carriage position.
211EH - 211FH	Check to see if the current carriage position in register A is greater than 112.
2120H - 2122H	Jump.
2123H - 2125H	Load register A with the video line length.
2126H	Load register B with the video line length in register A.
2127H - 2129H	Load register A with the current video line position.
212AH	Check to see if the current video line position in register A is equal to the video line length in register B.
212BH - 212DH	Go print a carriage return on the current output device if necessary.

212EH - 212FH	Jump if this is the end of the line on the current output device.
2130H - 2131H	Check to see if there is at least 16 spaces left on the current line for the current output device.
2132H - 2133H	Loop till there are at least 16 spaces left on the current line.
2134H	Figure the number of spaces to be sent to the current output device.
2135H - 2136H	Jump.
2137H - 2139H	Go evaluate the tab number at the location of the current BASIC program pointer in register pair HL and return with the result in register A.
213AH - 213BH	Mask the tab number in register A so that it doesn't exceed 63.
213CH	Load register B with the value of the tab number in register A.
213DH - 213EH	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a).
213FH	Decrement the value of the current BASIC program pointer in register pair HL.
2140H	Save the value of the current BASIC program pointer in register pair HL on the stack.
2141H - 2143H	Call the DOS link at 41DCH.
2144H - 2146H	Load register A with the value of the current output device flag.
2147H	Test the value of the current output device flag in register A.
2148H - 214AH	Go to the Level II BASIC error routine and display a FC ERROR message if the current output device is the cassette recorder.
214BH - 214DH	Jump if the current output device is the video display.
214EH - 2150H	Load register A with the current carriage position.
2151H - 2152H	Jump.
2154H - 2155H	Load register A with the current video line position.
2156H	Complement the current line position in register A.

2157H	Add the tab number in register B to the adjusted line position in register A.
2158H - 2159H	Jump if the tab number in register B is less than the line position in register A.
215AH	Bump the number of spaces to be printed in register A.
215BH	Load register B with the number of spaces to be printed in register A.
215CH - 215DH	Load register A with a space.
215EH - 2160H	Send the space in register A to the current output device.
2161H	Decrement the value of the space counter in register B.
2162H - 2163H	Loop until all of the spaces have been printed.
2164H	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
2165H	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
2166H - 2168H	Loop until done.
2169H - 216BH	Load register A with the current output device flag.
216CH	Test the value of the current output device flag in register A.
216DH - 216FH	If the current output device flag is the cassette recorder, then go turn it off.
2170H	Load register A with the video display output device code.
2171H - 2173H	Save the value in register A as the current value of the output device flag.
2174H - 2176H	Call the DOS link at 41BEH.
2177H	Return.
2178H - 217EH	**MESSAGE STORAGE LOCATION**
2178H - 217EH	The REDO message is stored here.
217FH - 2285H	**LEVEL II BASIC INPUT AND READ ROUTINES**
217FH - 2181H	Load register A with the read flag.

2182H	Check to see if the read flag is set.
2183H - 2185H	Jump to the Level II BASIC error routine if the read flag is set.
2186H - 2188H	Load register A with the input type flag.
2189H	Check to see if the cassette recorder is the current input device.
218AH - 218BH	Load register E with the FD ERROR code.
218CH - 218EH	Go to the Level II BASIC error routine and display a FD ERROR message if the current input device is the cassette recorder.
218FH	Clean up the stack.
2190H - 2192H	Load register pair HL with the starting address of the REDO message.
2193H - 2195H	Go display the REDO message.
2196H - 2198H	Get the value of the current BASIC program pointer in register pair HL.
2199H	Return.
219AH - 219CH	Check to see if there is an illegal direct in the input statement.
219DH	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
219EH - 21A0H	Call the DOS link at 41D6H.
21A1H - 21A2H	Check to see if the character at the location of the current BASIC program pointer in register A is a #.
21A3H - 21A5H	Set the current input device flag for the cassette recorder.
21A6H	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
21A7H - 21A8H	Jump if there is keyboard input.
21A9H - 21ABH	Go read the cassette leader and find the sync byte.
21ACH	Save the current BASIC program pointer in register pair HL on the stack.
21ADH - 21AEH	Load register B with the size of the input buffer.

21AFH - 21B1H Load register pair HL with the starting address of the input buffer.

21B2H - 21B4H Go read a byte from the cassette recorder and return with it in register A.

21B5H Save the byte read from the cassette recorder in register A at the location of the input buffer pointer in register pair HL.

21B6H Bump the value of the input buffer pointer in register pair HL.

21B7H - 21B8H Check to see if the character read from the cassette recorder in register A is a carriage return.

21B9H - 21BAH Jump if the character read from the cassette recorder in register A is a carriage return.

21BBH - 21BCH Loop till the input buffer is full.

21BDH Decrement the value of the input buffer pointer in register pair HL.

21BEH - 21BFH Save a zero at the location of the input buffer pointer in register pair HL.

21C0H - 21C2H Go turn the cassette recorder off.

21C3H - 21C5H Load register pair HL with the starting address of the input buffer.

21C6H Decrement the value of the input buffer pointer in register pair HL.

21C7H - 21C8H Jump.

21C9H - 21CBH Load register pair BC with the return address.

21CCH Save the value of the return address in register pair BC on the stack.

21CDH - 21CEH Check to see if the character at the location of the current BASIC program pointer in register A is a quote.

21CFH Return if the character at the location of the current BASIC program pointer in register A isn't a quote.

21D0H - 21D2H Go set up pointers for the prompting message in the temporary string work area.

21D3H - 21D4H Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a semicolon.

21D5H	Save the value of the current BASIC program pointer in register pair HL on the stack.
21D6H - 21D8H	Go display the prompting message.
21D9H	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
21DAH	Return.
21DBH	Save the value of the current BASIC program pointer in register pair HL on the stack.
21DCH - 21DEH	Go get the input from the keyboard.
21DFH	Get the value of the current BASIC program pointer from the stack and put it in register pair BC.
21E0H - 21E2H	Jump if the BREAK key was pressed.
21E3H	Bump the value of the input buffer pointer in register pair HL.
21E4H	Load register A with the character at the location of the input buffer pointer in register pair HL.
21E5H	Test the value of the character at the location of the input buffer pointer in register A.
21E6H	Decrement the value of the input buffer pointer in register pair HL.
21E7H	Save the value of the current BASIC program pointer in register pair BC on the stack.
21E8H - 21EAH	Jump if the character at the location of the input buffer pointer in register A is an end of the input character.
21EBH - 21ECH	Save a comma at the location of the current input buffer pointer in register pair HL.
21EDH - 21EEH	Jump.
21EFH	Save the current BASIC program pointer in register pair HL.
21F0H - 21F2H	Load register pair HL with the location of the last DATA statement read.
21F3H	Set register A to a nonzero value if entered from the READ routine.
21F4H	This sets register A to zero if entered from the INPUT routine.

21F5H - 21F7H	Save the value of the input type flag in register A.
21F8H	Exchange the start of the input pointer in register pair HL with the current BASIC program pointer on the stack.
21F9H - 21FAH	Jump.
21FBH - 21FCH	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a comma.
21FDH - 21FFH	Go get the address of the variable at the location of the current BASIC program pointer in register pair HL and return with it in register pair DE.
2200H	Exchange the value of the current BASIC program pointer in register pair HL with the start of the input buffer pointer on the stack.
2201H	Save the variable's address in register pair DE on the stack.
2202H	Load register A with the character at the location of the input buffer pointer in register pair HL.
2203H - 2204H	Check to see if the character at the location of the input buffer pointer register A is a comma.
2205H - 2206H	Jump if the character at the location of the input buffer pointer in register A is a comma.
2207H - 2209H	Load register A with the input type flag.
220AH	Check for READ or INPUT.
220BH - 220DH	Jump if the input type flag in register A indicates READ.
220EH - 2210H	Load register A with the value of the cassette input flag.
2211H	Check to see if the input is from the cassette recorder.
2212H - 2213H	Load register E with an OD ERROR code.
2214H - 2216H	Go to the Level II BASIC error routine and display an OD ERROR message if the input is from the cassette recorder.
2217H - 2218H	Load register A with a ?.
2219H - 221BH	Go display the ? in register A.
221CH - 221EH	Go get the keyboard input.

221FH	Get the address of the variable to be set from the stack and put it in register pair DE.
2220H	Get the value of the current BASIC program pointer from the stack and put it in register pair BC.
2221H - 2223H	Jump if the BREAK key was pressed.
2224H	Bump the value of the input buffer pointer in register pair HL.
2225H	Load register A with the character at the location of the input buffer pointer in register pair HL.
2226H	Check to see if the character at the location of the input buffer pointer in register A is an end of the input character.
2227H	Decrement the value of the input buffer pointer in register pair HL.
2228H	Save the value of the current BASIC program pointer in register pair BC on the stack.
2229H - 222BH	Jump if the character at the location of the input buffer pointer in register A is an end of the input character.
222CH	Save the variable's address in register pair DE on the stack.
222DH - 222FH	Call the DOS link at 41DCH.
2230H	Go check the current value of the number type flag.
2231H	Save the number type of the variable on the stack.
2232H - 2233H	Jump if the variable to be set is numeric.
2234H	Bump the value of the input buffer pointer in register pair HL.
2235H	Load register D with the character at the location of the input buffer pointer in register A.
2236H	Load register B with the character at the location of the input buffer pointer in register A.
2237H - 2238H	Check to see if the character at the location of the input buffer pointer in register A is a quote.
223AH - 223CH	Jump if the character at the location of the input buffer pointer in register A is a quote.
223DH - 223EH	Load register B with the scan character.

223FH	Decrement the value of the input buffer pointer in register pair HL.
2240H - 2242H	Go set up the pointers for the temporary string work area.
2243H	Get the number type for the variable from the stack and put it in register A.
2244H	Load register pair DE with the value of the input buffer pointer in register pair HL.
2245H - 2247H	Load register pair HL with the value of the return address.
2248H	Exchange the value of the return address in register pair HL with the value of the current BASIC program pointer on the stack.
2249H	Save the value of the input buffer pointer in register pair DE on the stack.
224AH - 224CH	Go set the variable to the value of the string.
224DH	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
224EH	Load register A with the number type for the variable to be set.
224FH	Save the value in register pair AF on the stack.
2250H - 2252H	Load register pair BC with the value of the return address.
2253H	Save the return address in register pair BC on the stack.
2254H - 2256H	Go convert the ASCII string at the location of the input buffer pointer in register pair HL if the current number type is integer or single precision.
2257H - 2259H	Go convert the ASCII string at the location of the input buffer pointer in register pair HL to binary if the current number type is double precision.
225AH	Decrement the value of the input buffer pointer in register pair HL.
225BH	Go bump the value of the input buffer pointer in register pair HL till it points to the next character.
225CH - 225DH	Jump if the character at the location of the input buffer pointer in register pair HL is an end of the input character.

225EH - 225FH	Check to see if the character at the location of the input buffer pointer in register A is a comma.
2260H - 2262H	Jump if the character at the location of the input buffer pointer in register A isn't a comma.
2263H	Exchange the value of the input buffer pointer in register pair HL with the value of the current BASIC program pointer on the stack.
2264H	Decrement the value of the current BASIC program pointer in register pair HL.
2265H	Bump the value of the current BASIC program pointer in register pair HL.
2266H - 2268H	Go examine the next variable if the character at the location of the current BASIC program pointer in register pair HL isn't an end of the BASIC statement character.
2269H	Clean up the stack.
226AH - 226CH	Load register A with the error flag.
226DH	Check to see if the error flag in register A indicates an error.
2263H	Return to the BASIC interpreter if the error flag in register A doesn't indicate an error.
226FH - 2271H	Load register A with the value of the input type flag.
2272H	Check to see if the input type is READ or INPUT.
2273H	Load register pair DE with the value of the current BASIC program pointer in register pair HL.
2274H - 2276H	Jump if the input type flag is set for READ.
2277H	Save the current BASIC program pointer in register pair DE on the stack.
2278H - 227AH	Call the DOS link at 41DFH.
227BH	Check to see if this is the end of the input.
227CH - 227EH	Load register pair HL with the starting address of the EXTRA IGNORED message.
227FH - 2281H	Go display the EXTRA IGNORED message.
2282H	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
2283H - 2285H	Go turn off the cassette recorder and return to the BASIC interpreter.

2286H - 2295H MESSAGE STORAGE LOCATION

2286H - 2295H The EXTRA IGNORED message is stored here.

2296H - 22B5H FIND THE NEXT DATA STATEMENT ROUTINE

2296H - 2298H Go find the next DATA statement.

2299H Check to see if this is the end of the BASIC line.

229AH - 229BH Jump if the BASIC statement is terminated with a :.

229CH Bump the value of the current BASIC program pointer in register pair HL.

229DH Load register A with the LSB of the line address at the location of the current BASIC program pointer in register pair HL.

229EH Bump the value of the current BASIC program pointer in register pair HL.

229FH Combine the MSB of the line address at the location of the current BASIC program in register pair HL with the LSB of the line address in register A.

22A0H - 22A1H Load register E with an OD ERROR code.

22A2H - 22A4H Go to the Level II BASIC error routine and display an OD ERROR message if this is the end of the BASIC program.

22A5H Bump the value of the current BASIC program pointer in register pair HL.

22A6H Load register E with the LSB of the BASIC line number at the location of the current BASIC program pointer in register pair HL.

22A7H Bump the value of the current BASIC program pointer in register pair HL.

22A8H Load register D with the MSB of the BASIC line number at the location of the current BASIC program pointer in register pair HL.

22A9H Exchange the value of the current BASIC program pointer in register pair HL with the value of the BASIC line number in register pair DE.

22AAH - 22ACH Save the BASIC line number in register pair HL.

22ADH Exchange the value of the current BASIC program pointer in register pair DE with the value of the BASIC line number in register pair HL.

22AEH	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
22AFH - 22B0H	Check to see if the character at the location of the current BASIC program pointer in register A is a DATA token.
22B1H - 22B2H	Jump if the character at the location of the current BASIC program pointer in register A isn't a DATA token.
22B3H - 22B5H	Jump.
22B6H - 2336H	**LEVEL II BASIC NEXT ROUTINE**
22B6H - 22B8H	Load register pair DE with the default variable address.
22B9H - 22BBH	Go examine the variable if one follows the NEXT token.
22BCH - 22BEH	Save the value of the current BASIC program pointer in register pair HL.
22BFH - 22C1H	Go search the stack for the appropriate FOR push.
22C2H - 22C4H	Go to the Level II BASIC error routine and display a NF ERROR message if the appropriate FOR push wasn't found.
22C5H	Load the stack pointer with the value of the memory pointer in register pair HL.
22C6H - 22C8H	Save the value in register pair HL.
22C9H	Save the variable's address in register pair DE on the stack.
22CAH	Load register A with the value of the sign for the STEP value.
22CBH	Bump the value of the memory pointer in register pair HL.
22CCH	Save the value of the sign for the STEP value in register A on the stack.
22CDH	Save the variable's address in register pair DE on the stack.
22CEH	Load register A with the number type flag for the STEP value.
22CFH	Bump the value of the memory pointer in register pair HL.

22D0H	Check the value of the number type flag for the STEP flag in register A.
22D1H - 22D3H	Jump if the STEP value is an integer.
22D4H - 22D6H	Go move the single precision value at the location of the memory pointer in register pair HL into register pairs BC and DE.
22D7H	Exchange the value of the variable's address on the stack with the value of the memory pointer in register pair HL.
22D8H	Save the value of the variable's address in register pair HL on the stack.
22D9H - 22DBH	Go add the single precision value at the location of the memory pointer in register pair HL to the single precision STEP value in register pairs BC and DE. Return with the result in REG1.
22DCH	Get the value of the variable's address from the stack and put it in register pair HL.
22DDH - 22DFH	Go move the single precision result from REG1 to the variable's address in register pair HL.
22E0H	Get the value of the memory pointer from the stack and put it in register pair HL.
22E1H - 22E3H	Go move the single precision TO value at the location of the memory pointer in register pair HL into register pairs BC and DE.
22E4H	Save the value of the memory pointer in register pair HL on the stack.
22E5H - 22E7H	Go compare the single precision result in REG1 to the single precision TO value in register pairs BC and DE.
22E8H - 22E9H	Jump.
22EAH	Bump the value of the memory pointer in register pair HL.
22EBH	Bump the value of the memory pointer in register pair HL.
22ECH	Bump the value of the memory pointer in register pair HL.
22EDH	Bump the value of the memory pointer in register pair HL.
22EEH	Load register C with the LSB of the STEP value at

	the location of the memory pointer in register pair HL.
22EFH	Bump the value of the memory pointer in register pair HL.
22F0H	Load register B with the MSB of the STEP value at the location of the memory pointer in register pair HL.
22F1H	Bump the value of the memory pointer in register pair HL.
22F2H	Exchange the value of the variable's address on the stack with the value of the memory pointer in register pair HL.
22F3H	Load register E with the LSB of the integer variable at the location of the memory pointer in register pair HL.
22F4H	Bump the value of the memory pointer in register pair HL.
22F5H	Load register D with the MSB of the integer variable at the location of the memory pointer in register pair HL.
22F6H	Save the value of the memory pointer in register pair HL on the stack.
22F7H	Load register L with the LSB of the STEP value in register C.
22F8H	Load register H with the MSB of the STEP value in register B.
22F9H - 22FBH	Go add the STEP value in register pair HL to the integer value in register pair DE.
22FCH - 22FEH	Load register A with the current value of the number type flag.
22FFH	Check to see if the current value in REG1 is single precision.
2301H - 2303H	Go to the Level II BASIC error routine and display an OV ERROR message if the current value in REG1 is single precision.
2304H	Load register pair DE with the integer result in register pair HL.
2305H	Get the value of the memory pointer from the stack and put it in register pair HL.

2306H	Save the MSB of the result in register D at the location of the memory pointer in register pair HL.
2307H	Decrement the value of the memory pointer in register pair HL.
2308H	Save the LSB of the result in register E at the location of the memory pointer in register pair HL.
2309H	Get the value of the memory pointer from the stack and put it in register pair HL.
230AH	Save the integer value in register pair DE on the stack.
230BH	Load register E with the LSB of the TO value at the location of the memory pointer in register pair HL.
230CH	Bump the value of the memory pointer in register pair HL.
230DH	Load register D with the MSB of the TO value at the location of the memory pointer in register pair HL.
230EH	Bump the value of the memory pointer in register pair HL.
230FH	Exchange the integer value on the stack with the value of the memory pointer in register pair HL.
2310H - 2312H	Go compare the integer result in register pair HL with the TO value in register pair DE.
2313H	Get the value of the memory pointer from the stack and put it in register pair HL.
2314H	Get the value of the sign from the stack and put it in register pair BC.
2315H	Subtract the value of the sign in register B from the value in register A.
2316H - 2318H	Go load register pair BC with the FOR statement's BASIC program pointer.
2319H - 231AH	Jump if the FOR. .NEXT loop has been completed.
231BH	Load register pair HL with the BASIC line number in register pair DE.
231CH - 231EH	Save the BASIC line number in register pair HL as the current BASIC line number.
231FH	Load register L with the LSB of the current BASIC program pointer in register C.

2320H	Load register H with the MSB of the current BASIC program pointer in register B.
2321H - 2323H	Jump.
2324H	Load the stack pointer with the value of the memory pointer in register pair HL.
2325H - 2327H	Load register pair HL with the value of the current BASIC program pointer.
2328H - 232AH	Save the value of the current BASIC program pointer in register pair HL.
232BH	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
232CH - 232DH	Check to see if the character at the location of the current BASIC program pointer in register A is a comma.
232EH - 2330H	Jump if the character at the location of the current BASIC program pointer in register A isn't a comma.
2331H	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
2332H - 2334H	Go do NEXT.
2335H - 2336H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a left parenthesis.
2337H - 27C8H	**EVALUATE EXPRESSION**
2337H	Decrement the value of the current BASIC program pointer in register pair HL.
2338H - 2339H	Load register D with zero.
233AH	Save the value in register pair DE on the stack.
233BH - 233CH	Load register C with the number of bytes of memory required.
233DH - 233FH	Go do a memory check.
2340H - 2342H	Go get the value of the next part of the expression at the location of the current BASIC program pointer in register pair HL.
2343H - 2345H	Save the value of the current BASIC program pointer in register pair HL.

2346H - 2348H	Load register pair HL with the value of the current BASIC program pointer.
2349H	Get the last precedence value from the stack and put it in register pair BC.
234AH	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
234BH - 234CH	Load register D with zero.
234DH - 234EH	Check to see if the character at the location of the current BASIC program pointer in register A is an arithmetic or logical token.
234FH - 2350H	Jump if the character at the location of the current BASIC program pointer in register A is an arithmetic or logical token.
2351H - 2352H	Check to see if the character at the location of the current BASIC program pointer in register A is a greater than, less than, or equal to token.
2353H - 2354H	Jump if the character at the location of the current BASIC program pointer in register A isn't a greater than, less than, or equal to token.
2355H - 2356H	Set the Carry flag according to the token at the location of the current BASIC program pointer in register A.
2357H	Adjust the value in register A to indicate greater than, less than, or equal to.
2358H	Combine the current value in register A with the value of the last token examined to see if this is an illegal combination of operators.
2359H	Check for an illegal combination of operators.
235AH	Load register D with the value in register A.
235BH - 235DH	Go to the Level II BASIC error routine and display a SN ERROR message if this is an illegal combination of operators.
235EH - 2360H	Save the value of the current BASIC program pointer in register pair HL.
2361H	Bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
2362H - 2363H	Jump.

2364H	Load register A with the value of the operator in register D.
2365H	Test the value of the operator in register A.
2366H - 2368H	Jump if the operator in register A is a greater than, less than, or equal to.
2369H	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
236AH - 236CH	Save the address of the current BASIC program pointer in register pair HL.
236DH - 236EH	Check to see if the operator at the location of the current BASIC program pointer in register A is an arithmetic token.
236FH	Return if the character at the location of the current BASIC program pointer in register A isn't an arithmetic token.
2370H - 2371H	Check to see if the character at the location of the current BASIC program pointer in register A is a + to OR token.
2372H	Return if the character at the location of the current BASIC program pointer in register A isn't a + to OR token.
2373H	Load register E with the operator value in register A.
2374H - 2376H	Load register A with the current value of the number type flag.
2377H - 2378H	Adjust the value of the number type flag.
2379H	Combine the operator value in register E with the adjusted number type flag in register A.
237AH - 237CH	Jump if the combination of the adjusted number type flag in register A and the operator value in register E indicates string addition.
237DH - 237FH	Load register pair HL with the starting address of the table of precedence values.
2380H	Add the value of the operator in register pair DE to the table of precedence values pointer in register pair HL.
2381H	Load register A with the precedence value for the last operator in register B.

2382H		Load register D with the precedence value for the current operator.
2383H		Compare the precedence value for the current operator in register D with the precedence value for the last operator in register A.
2384H		Return if the precedence value for the current operator in register D is greater than the precedence value for the last operator in register A.
2385H		Save the precedence value and the token for the last operator in register pair BC on the stack.
2386H	- 2388H	Load register pair BC with the return address.
2389H		Save the return address in register pair BC on the stack.
238AH		Load register A with the precedence value for the current operator in register D.
238BH	- 238CH	Check to see if the precedence value for the current operator in register A indicates an exponential operator.
238DH	- 238FH	Jump if the precedence value for the current operator in register A indicates an exponential operator.
2390H	- 2391H	Check to see if the precedence value for the current operator in register A indicates a logical operator.
2392H	- 2394H	Jump if the precedence value for the current operator in register A indicates a logical operator.
2395H	- 2397H	Load register pair HL with the address of REG1.
2398H		Clear the flags.
2399H	- 239BH	Load register A with the current value of the number type flag.
239CH		Adjust the current number type flag in register A.
239DH		Adjust the current number type flag in register A.
239EH		Adjust the current number type flag in register A.
239FH	- 23A1H	Go to the Level II BASIC error routine and display a TM ERROR message if the current value in REG1 is a string.
23A2H		Get the value at the location of the memory pointer in register pair HL and put it in register C.

23A3H	Bump the value of the memory pointer in register pair HL.
23A4H	Load register B with the value at the location of the memory pointer in register pair HL.
23A5H	Save the value in register pair BC on the stack.
23A6H - 23A8H	Jump if the current number type is an integer.
23A9H	Bump the value of the memory pointer in register pair HL.
23AAH	Load register C with the value at the location of the memory pointer in register pair HL.
23ABH	Bump the value of the memory pointer in register pair HL.
23ACH	Load register B with the value at the location of the memory pointer in register pair HL.
23ADH	Save the value in register pair BC on the stack.
23AEH	Save the value in register pair AF on the stack.
23AFH	Check to see if the current number type is double precision.
23B0H - 23B2H	Jump if the current number type is single precision.
23B3H	Get the value from the stack and put it in register pair AF.
23B4H	Bump the value of the memory pointer in register pair HL.
23B5H - 23B6H	Jump if the current number type is single precision.
23B7H - 23B9H	Load register pair HL with the starting address of REG1.
23BAH	Load register C with the value at the location of the memory pointer in register pair HL.
23BBH	Bump the value of the memory pointer in register pair HL.
23BCH	Load register B with the value at the location of the memory pointer in register pair HL.
23BDH	Bump the value of the memory pointer in register pair HL.
23BEH	Save the value in register pair BC on the stack.

23BFH	Load register C with the value at the location of the memory pointer in register pair HL.
23C0H	Bump the value of the memory pointer in register pair HL.
23C1H	Load register B with the value at the location of the memory pointer in register pair HL.
23C2H	Save the value in register pair BC on the stack.
23C4H	Get the value from the stack and put it in register pair AF.
23C5H - 23C6H	Adjust the number type in register A.
23C7H	Load register C with the value of the current operator token in register E.
23C8H	Load register B with the number type flag in register A.
23C9H	Save the value in register pair BC on the stack.
23CAH - 23CCH	Load register pair BC with the return address.
23CDH	Save the return address in register pair BC on the stack.
23CEH - 23D0H	Load register pair HL with the value of the current BASIC program pointer.
23D1H - 23D3H	Jump.
23D4H - 23D6H	Go convert the integer result in REG1 to single precision.
23D7H - 23D9H	Go move the single precision result in REG1 onto the stack.
23DAH - 23DCH	Load register pair BC with the address of the exponential routine.
23DDH - 23DEH	Load register D with the precedence value for an exponential operator.
23DFH - 23E0H	Jump.
23E1H	Save the precedence value and the operator token in register pair DE on the stack.
23E2H - 23E4H	Go convert the current result in REG1 to an integer.
23E5H	Get the precedence value and the operator token from the stack and put it in register pair DE.

23E6H	Save the integer value in register pair HL on the stack.
23E7H - 23E9H	Load register pair BC with the return address.
23EAH - 23EBH	Jump.
23ECH	Load register A with the precedence value for the last operator in register B.
23EDH - 23EEH	Check to see if the last operator was a logical operator.
23EFH	Jump if the last operator was a logical operator.
23F0H	Save the precedence value and the operator token for the last operator in register pair BC on the stack.
23F1H	Save the precedence value and the token for the current operator in register pair DE on the stack.
23F2H - 23F4H	Load register pair DE with the precedence value and the token for the new operator.
23F5H - 23F7H	Load register pair HL with the address to perform the current operation.
23F8H	Save the value of the address in register pair HL on the stack.
23F9H	Go check the value of the current number type flag.
23FAH - 23FCH	Jump if the current value in REG1 is numeric.
23FDH - 23FFH	Load register pair HL with the string address in REG1.
2400H	Save the string's address in register pair HL on the stack.
2401H - 2303H	Load register pair BC with the address of the string comparison routine.
2404H - 2405H	Jump.
2406H	Get the precedence value and the token for the last operator from the stack and put it in register pair BC.
2407H	Load register A with the operator token in register C.
2408H - 240AH	Save the operator token in register A.
240BH	Load register A with the precedence value in register B.

240CH - 240DH	Check the number type for the precedence value in register A.
240EH - 240FH	Jump if the number type in register A is double precision.
2410H - 2412H	Load register A with the number type for the current result in REG1.
2413H - 2414H	Check to see if the current result in REG1 is double precision.
2415H - 2417H	Jump if the current value in REG1 is double precision.
2418H	Load register D with the current number type flag in register A.
2419H	Load register A with the precedence value for the last operator in register B.
231AH - 241BH	Check to see if the number type for the precedence value in register A is single precision.
241CH - 241EH	Jump if the number type for the precedence value in register A is single precision.
241FH	Load register A with the number type flag for the value in REG1.
2420H - 2421H	Check to see if the current value in REG1 is a string.
2422H - 2424H	Go to the Level II BASIC error routine and display a TM ERROR message if the current value in REG1 is a string.
2325H - 2427H	Jump if the current value in REG1 is single precision.
2428H - 242AH	Load register pair HL with the starting address of the arithmetic jump table.
242BH - 242CH	Load register B with zero.
242DH	Add the operator's token in register pair BC to the value of the arithmetic jump table pointer in register pair HL.
242EH	Add the operator's token in register pair BC to the value of the arithmetic jump table pointer in register pair HL.
242FH	Load register C with the LSB of the jump address at the location of the arithmetic jump table pointer in register pair HL.

2430H	Bump the value of the arithmetic jump table pointer in register pair HL.
2431H	Load register B with the MSB of the jump address at the location of the arithmetic jump table pointer in register pair HL.
2432H	Get the value for the previous result from the stack and put it in register pair DE.
2433H - 2435H	Get the value for the current result from REG1 and put it in register pair HL.
2436H	Save the arithmetic routine's jump address in register pair BC on the stack.
2437H	Return.
2438H - 243AH	Go convert the result in REG1 to double precision.
243BH - 243DH	Go convert the current result in REG1 to single precision.
243EH	Get the value from the stack and put it in register pair HL.
243FH - 2441H	Save the value in register pair HL in REG1.
2442H	Get the value from the stack and put it in register pair HL.
2443H - 2445H	Save the value in register pair HL in REG1.
2446H	Get the value from the stack and put it in register pair BC.
2447H	Get the value from the stack and put it in register pair DE.
2448H - 244AH	Go move the 32-bit value in register pairs BC and DE into REG1.
244BH - 244DH	Go convert the previous value to double precision.
244EH - 2450H	Load register pair HL with the double precision arithmetic jump table's starting address.
2451H - 2453H	Load register A with the operator token in register A.
2454H	Multiply the value of the operator token in register A by two.
2455H	Save the value in register pair BC on the stack.
2456H	Load register C with the adjusted value of the operator token in register A.

2457H - 2458H	Load register B with zero.
2459H	Add the value of the adjusted operator token in register pair BC to the value of the double precision arithmetic jump table pointer in register pair HL.
245AH	Get the value from the stack and put it in register pair BC.
245BH	Load register A with the LSB of the jump address at the location of the arithmetic jump table pointer in register pair HL.
245CH	Bump the value of the arithmetic jump table pointer in register pair HL.
245DH	Load register H with the MSB of the jump address at the location of the arithmetic jump table pointer in register pair HL.
245EH	Load register L with the LSB of the jump address in register A.
245FH	Jump to the arithmetic routine whose address is in register pair HL.
2460H	Save the precedence value and the operator token in register pair BC on the stack.
2461H - 2463H	Move the current result to the a storage area.
2464H	Load register A with the value of the current number type flag.
2465H - 2467H	Save the value of the current number type flag in register A.
2468H - 2469H	Check to see if the current result in REG1 is single precision.
246AH - 246BH	Jump if the current result in REG1 is single precision.
246CH	Get the integer value from the stack and put it in register pair HL.
246DH - 246FH	Save the integer value in register pair HL in REG1.
2470H - 2471H	Jump.
2472H - 2474H	Go convert the current result in REG1 to single precision.
2475H	Get the value from the stack and put it in register pair BC.

2476H	Get the value from the stack and put it in register pair DE.
2477H - 2479H	Load register pair HL with the starting address of the single precision arithmetic jump table.
247AH - 247BH	Jump.
247CH	Get the integer value from the stack and put it in register pair HL.
247DH - 247FH	Save the current single precision result in REG1 on the stack.
2480H - 2482H	Go convert the integer value in register pair HL to single precision.
2483H - 2485H	Save the single precision value in REG1 in register pairs BC and DE.
2486H	Get the value from the stack and put it in register pair HL.
2487H - 2489H	Save the value in register pair HL in REG1.
248AH	Get the value from the stack and put it in register pair HL.
248BH - 248DH	Save the value in register pair HL in REG1.
248EH - 248FH	Jump.
2490H	Save the integer value in register pair HL on the stack.
2491H	Load register pair HL with the integer value in register pair DE.
2492H - 2494H	Go convert the integer value in register pair HL to single precision and return with the result in REG1.
2495H	Get the integer value from the stack and put it in register pair HL.
2496H - 2498H	Go move the single precision value in REG1 on to the stack.
2499H - 249BH	Go convert the integer value in register pair HL to single precision and return with the result in REG1.
249CH - 249EH	Go do a single precision divide.
249FH	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.

24A0H - 24A1H Load register E with a MO ERROR code.

24A2H - 24A4H Go to the Level II BASIC error routine and display a MO ERROR message if this is the end of the expression.

24A5H - 24A7H Jump if the character at the location of the current BASIC program pointer in register pair HL is numeric.

24A8H - 24AAH Check to see if the character at the location of the current BASIC program pointer in register pair HL is alphabetic.

24ABH - 24ADH Jump if the character at the location of the current BASIC program pointer in register pair HL is alphabetic.

24AEH - 24AFH Check to see if the character at the location of the current BASIC program pointer in register A is a + token.

24B0H - 24B1H Jump if the character at the location of the current BASIC program pointer in register A is a + token.

24B2H - 24B3H Check to see if the character at the location of the current BASIC program pointer in register A is a decimal point.

24B4H - 24B6H Jump if the character at the location of the current BASIC program pointer in register A is a decimal point.

24B7H - 24B8H Check to see if the character at the location of the current BASIC program pointer in register A is a – token.

24B9H - 24BBH Jump if the character at the location of the current BASIC program pointer in register A is a – token.

24BCH - 24BDH Check to see if the character at the location of the current BASIC program pointer in register A is a quote.

24BEH - 24C0H Jump if the character at the location of the current BASIC program pointer in register A is a quote.

24C1H - 24C2H Check to see if the character at the location of the current BASIC program pointer in register A is a NOT token.

24C3H - 24C5H Jump if the character at the location of the current BASIC program pointer in register A is a NOT token.

24C6H - 24C7H Check to see if the character at the location of the current BASIC program pointer in register A is a &.

24C8H - 24CAH Jump if the character at the location of the current BASIC program pointer in register A is a &.

24CBH - 24CCH Check to see if the character at the location of the current BASIC program pointer in register A is an ERR token.

24CDH - 24CEH Jump if the character at the location of the current BASIC program pointer in register A isn't an ERR token.

24CFH Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.

24D0H - 24D2H Load register A with the value of the current error number.

24D3H Save the value of the current BASIC program pointer in register pair HL on the stack.

24D4H - 24D6H Go save the value of the current error number in register A as the current result in REG1.

24D7H Get the value of the current BASIC program pointer from the stack and put it in register pair HL.

24D8H Return.

24D9H - 24DAH Check to see if the character at the location of the current BASIC program pointer in register A is a ERL token.

24DBH - 24DCH Jump if the character at the location of the current BASIC program pointer in register A isn't an ERL token.

24DDH Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.

24DEH Save the value of the current BASIC program pointer in register pair HL on the stack.

24DFH - 24E1H Load register pair HL with the current error line number.

24E2H - 24E3H Go save the error line number in register pair HL as the current result in REG1.

24E5H Get the value of the current BASIC program pointer from the stack and put it in register pair HL.

24E6H Return.

24E7H - 24E8H Check to see if the character at the location of the

current BASIC program pointer in register A is a VARPTR token.

24E9H - 24EAH Jump if the character at the location of the current BASIC program pointer in register A isn't a VARPTR token.

24EBH Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.

24ECH - 24EDH Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a (.

24EEH - 24F0H Go evaluate the variable name at the location of the current BASIC program pointer in register pair HL.

24F1H - 24F2H Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a).

24F3H Save the value of the current BASIC program pointer in register pair HL on the stack.

24F4H Load register pair HL with the variable's address in register pair DE.

24F5H Load register A with MSB of the variable's address in register H.

24F6H Combine the LSB of the variable's address in register L with the MSB of the variable's address in register A.

24F7H - 24F9H Go to the Level II BASIC error routine and display a FC ERROR message if the variable's address in register pair HL is equal to zero.

24FAH - 24FCH Save the variable's address in register pair HL as the current result in REG1.

24FDH Get the value of the current BASIC program pointer from the stack and put it in register pair HL.

24FEH Return.

24FFH - 2500H Check to see if the character at the location of the current BASIC program pointer in register A is a USR token.

2501H - 2503H Jump if the character at the location of the current BASIC program pointer in register A is a USR token.

2504H - 2505H Check to see if the character at the location of the

	current BASIC program pointer in register A is a INSTR token.
2506H - 2508H	Jump if the character at the location of the current BASIC program pointer in register A is an INSTR token.
2509H - 250AH	Check to see if the character at the location of the current BASIC program pointer in register A is a MEM token.
250BH - 250DH	Jump if the character at the location of the current BASIC program pointer in register A is a MEM token.
250EH - 250FH	Check to see if the character at the location of the current BASIC program pointer in register A is a TIME$ token.
2510H - 2512H	Jump if the character at the location of the current BASIC program pointer in register A is a TIME$ token.
2513H - 2514H	Check to see if the character at the location of the current BASIC program pointer in register A is a POINT token.
2515H - 2517H	Jump if the character at the location of the current BASIC program pointer in register A is a POINT token.
2518H - 2519H	Check to see if the character at the location of the current BASIC program pointer in register A is an INKEY$ token.
251AH - 251CH	Jump if the character at the location of the current BASIC program pointer in register A is an INKEY$ token.
251DH - 251EH	Check to see if the character at the location of the current BASIC program pointer in register A is a STRING$ token.
241FH - 2421H	Jump if the character at the location of the current BASIC program pointer in register A is a STRING$ token.
2522H - 2523H	Check to see if the character at the location of the current BASIC program pointer in register A is a FN token.
2524H - 2526H	Jump if the character at the location of the current BASIC program pointer in register A is a FN token.
2527H - 2528H	Check to see if the character at the location of the current BASIC program pointer in register A is a SGN to MID$ token.

2529H - 252BH	Jump if the character at the location of the current BASIC program pointer in register A is a SGN to MID$ token.
252CH - 252EH	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL.
242FH - 2530H	Check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a).
2431H	Return.
2532H	Load register D with the precedence value.
2534H - 2536H	Go evaluate the variable at the location of the current BASIC program pointer in register pair HL.
2537H - 2539H	Load register pair HL with the value of the current BASIC program pointer.
253AH	Save the value of the current BASIC program pointer in register pair HL on the stack.
253BH - 253DH	Invert the sign of the current value in REG1.
253EH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
253FH	Return.
2540H - 2542H	Get the address of the variable at the location of the current BASIC program pointer in register pair HL and return with it in register pair DE.
2543H	Save the value of the current BASIC program pointer in register pair HL on the stack.
2544H	Load register pair HL with the variable's address in register pair DE.
2545H - 2547H	Save the variable's address in register pair HL in REG1.
2548H	Go check the current value of the number type flag.
2549H - 254BH	Go move data if the current number type flag indicates a numeric value in REG1.
254CH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
254DH	Return.
254EH - 254FH	Load register B with zero.

2550H	Multiply the value of the operator token in register A by two.
2551H	Load register C with the adjusted value of the operator token in register A.
2552H	Save the value of the token in register pair BC on the stack.
2553H	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
2554H	Load register A with the value of the operator token in register C.
2555H - 2556H	Check the value of the operator token in register C.
2557H - 2558H	Jump if the value of the operator token in register A is SGN to CHR$.
2559H - 255BH	Go evaluate the expression.
255CH - 255DH	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a comma.
255EH - 2560H	Go display an error message if the current result isn't a string.
2561H	Exchange the address of the string in register pair DE with the value of the current BASIC program pointer in register pair HL.
2562H - 2564H	Load register pair HL with the VARPTR for the string.
2565H	Exchange the string's VARPTR in register pair HL with the value on the stack.
2566H	Save the value in register pair HL on the stack.
2567H	Load register pair HL with the value of the current BASIC program pointer in register pair DE.
2568H - 256AH	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL.
256BH	Exchange the integer value in register pair DE with the value of the current BASIC program pointer in register pair HL.
256CH	Exchange the integer value in register pair HL with the value on the stack.
256DH - 256EH	Jump.

256FH - 2571H	Evaluate the expression at the location of the current BASIC program pointer in register pair HL.
2572H	Exchange the value of the current BASIC program pointer in register pair HL with the operator value on the stack.
2573H	Load register A with the operator token in register L.
2574H - 2575H	Check to see if the operator token in register A is SGN to SQR.
2576H - 2577H	Jump if the operator token in register A is SGN to SQR.
2578H - 2579H	Check the value of the operator token in register A.
257AH	Save the operator value in register pair HL on the stack.
257BH - 257DH	Go convert the result in REG1 to single precision if the operator token in register A is SQR to ATN.
257EH	Get the operator value from the stack and put it in register pair HL.
257FH - 2581H	Load register pair DE with the value of the return address.
2582H	Save the value of the return address in register pair DE on the stack.
2583H - 2585H	Load register pair BC with the SGN to MID$ jump table address.
2586H	Add the jump table pointer in register pair BC with the value of the operator token in register pair HL.
2587H	Load register C with the LSB of the jump address at the location of the jump table pointer in register pair HL.
2588H	Bump the value of the jump table pointer in register pair HL.
2589H	Load register H with the MSB of the jump address at the location of the jump table pointer in register pair HL.
258AH	Load register L with the LSB of the jump address in register C.
258BH	Jump to the address in register pair HL.
258CH - 258EH	Go check to see if there is enough memory for the string.

258FH	Load register A with the length of the string at the location of the string's VARPTR in register pair HL.
2590H	Bump the value of the string's VARPTR in register pair HL.
2591H	Load register C with the LSB of the string's address at the location of the string's VARPTR in register pair HL.
2592H	Bump the value of the string's VARPTR in register pair HL.
2594H	Load register B with the MSB of the string's address at the location of the string's VARPTR in register pair HL.
2595H	Save the value of the string's address in register pair BC on the stack.
2596H	Save the string's length in register A on the stack.
2597H	Load register pair HL with the second string's VARPTR.
259AH	Get the length of the first string from the stack and put it in register D.
259BH	Load register E with the second string's length at the location of the second string's VARPTR in register pair HL.
259CH	Bump the value of the second string's VARPTR in register pair HL.
259DH	Load register C with the LSB of the second string's address at the location of the second string's VARPTR in register pair HL.
259EH	Bump the value of the second string's VARPTR in register pair HL.
259FH	Load register B with the MSB of the second string's address at the location of the second string's VARPTR in register pair HL.
25A0H	Get the first string's address from the stack and put it in register pair HL.
25A1H	Load register A with the length of the second string in register E.
25A2H	Combine the first string's length in register D with the second string's length in register A.
25A3H	Return if there aren't any characters left in either string to be compared.

25A4H	Load register A with the first string's length in register D.
25A5H - 25A6H	Check to see if the first string's length in register A is equal to zero.
25A7H	Return if there are no more characters in the first string to be compared.
25A8H	Load register A with zero.
25A9H	Check to see if the second string's length in register E is equal to zero.
25AAH	Bump the value in register A.
25ABH	Return if there aren't anymore characters in the second string to be compared.
25ACH	Decrement the value of the first string's length in register D.
25ADH	Decrement the value of the second string's length in register E.
25AEH	Load register A with the character at the location of the second string pointer in register pair BC.
25AFH	Compare the character at the location of the second string pointer in register A with the character at the location of the first string pointer in register pair HL.
25B0H	Bump the value of the first string pointer in register pair HL.
25B1H	Bump the value of the second string pointer in register pair BC.
25B2H - 25B3H	Jump if the characters match.
25B4H	Compliment the value of the Carry flag.
25B5H - 25B7H	Jump.
25B8H	Bump the value of the current precedence value in register A.
25B9H	Adjust the value of the current precedence value in register A.
25BAH	Get the last operator value from the stack and put it in register pair BC.
25BBH	Combine the precedence value in register B with the precedence value in register A.

25BCH - 25BDH	Adjust the value in register A.
25BEH	Check to see if the precedence values in registers A and B match.
25BFH - 25C1H	If the values match, set the current result in zero. If they do not match, set the current result to −1.
25C2H - 25C3H	Jump.
25C4H - 25C5H	Load register D with the precedence value.
25C6H - 25C8H	Go evaluate the expression.
25C9H - 25CBH	Go convert the current result in REG1 to an integer.
25CCH	Load register A with the LSB of the integer value.
25CDH	Compliment the LSB of the integer value in register A.
25CEH	Load register L with the adjusted LSB of the integer value in register A.
25CFH	Load register A with the MSB of the integer value in register H.
25D0H	Compliment the MSB of the integer value in register A.
25D1H	Load register H with the adjusted MSB of the integer value in register A.
25D2H - 25D4H	Save the complimented integer value in register pair HL as the current result in REG1.
25D5H	Clean up the stack.
25D6H - 25D8H	Jump.
25D9H - 25DBH	Load register A with the current value of the number type flag.
25DCH - 25DDH	Check to see if the current value in REG1 is double precision.
25DEH - 25DFH	Jump if the current value in REG1 is double precision.
25E0H - 25E1H	Adjust the value of the current number type flag in register A.
25E2H	Check the value of the adjusted number type flag in register A.
24E3H	Set the Carry flag.

25E4H	Return.
25E5H - 25E6H	Adjust the value of the current number type flag in register A.
25E7H	Test the value of the current number type flag in register A.
25E8H	Return.
25E9H	Save the operator value in register pair BC on the stack.
25EAH - 25ECH	Go convert the current result in REG1 to an integer.
25EDH	Get the operator value from the stack and put it in register A.
25EEH	Get the return address from the stack and put it in register pair DE.
25EFH - 25F1H	Load register pair BC with the return address.
25F2H	Save the value of the return address in register pair BC on the stack.
25F3H - 25F4H	Check to see if the operator value in register A is an OR token.
25F5H - 25F6H	Jump if the operator value in register A isn't an OR token.
25F7H	Load register A with the LSB of the first value in register E.
25F8H	Combine the LSB of the first value in register A with the LSB of the second value in register L.
25F9H	Load register L with the ORed value in register A.
25FAH	Load register A with the MSB of the second value in register H.
25FBH	Combine the MSB of the first value in register D with the MSB of the second value in register A.
25FCH	Return.
25FDH	Load register A with the LSB of the first value in register E.
25FEH	Combine the LSB of the first value in register A with the LSB of the second value in register L.
25FFH	Load register L with the ANDed value in register A.

2600H	Load register A with the MSB of the second value in register H.
2601H	Combine the MSB of the first value in register D with the MSB of the second value in register A.
2602H	Return.
2603H	Decrement the value of the BASIC program pointer in register pair HL.
2604H	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
2605H	Return if this is the end of the BASIC statement.
2606H - 2607H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a comma.
2608H - 260AH	Load register pair BC with the return address.
260BH	Save the value of the return address in register pair BC on the stack.
260CH - 260DH	Reset the value in register A.
260EH - 2610H	Save the value in register A as the current variable location/creation flag.
2611H	Load register B with the first character of the variable name.
2612H - 2614H	Make sure the first character of the variable name is a letter.
2615H - 2617H	Go to the Level II BASIC error routine and display a SN ERROR message if the first character of the variable name isn't a letter.
2618H	Zero register A.
2619H	Zero register C.
261AH	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
261BH - 261CH	Jump if the character at the location of the current BASIC program pointer in register A is numeric.
261DH - 261FH	Go check to see if the character at the location of the current BASIC program pointer is a letter.
2620H - 2621H	Jump if the character at the location of the current BASIC program pointer in register A isn't a letter.

2622H	Load register C with the second character of the variable name in register A.
2623H	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
2624H - 2625H	Loop till a nonnumeric character is found.
2626H - 2628H	Go check to see if the character at the location of the current BASIC program pointer in register pair HL is alphabetic.
2629H - 262AH	Jump if the character at the location of the current BASIC program pointer in register pair HL is alphabetic.
262BH - 262DH	Load register pair DE with the return address.
262EH	Save the value of the return address in register pair DE on the stack.
262FH - 2630H	Load register D with an integer number type flag.
2631H - 2632H	Check to see if the character at the location of the current BASIC program pointer in register A is a %.
2633H	Return if the character at the location of the current BASIC program pointer in register A is a %.
2634H	Bump register D so that it will be equal to a string number type flag.
2635H - 2636H	Check to see if the character at the location of the current BASIC program pointer in register A is a $.
2637H	Return if the character at the location of current BASIC program pointer in register A is a $.
2638H	Bump register D so that it will be equal to a single precision number type flag.
2639H - 263AH	Check to see if the character at the location of the current BASIC program pointer in register A is a !.
263BH	Return if the character at the location of the current BASIC program pointer in register A is a !.
263CH - 263DH	Load register D with a double precision number type flag.
263EH - 263FH	Check to see if the character at the location of the current BASIC program pointer in register A is a #.
2640H	Return if the character at the location of the current BASIC program pointer in register A is a #.

2641H	Load register A with the first character of the variable name in register B.
2642H - 2643H	Adjust the value of the first character of the variable name in register A so that it is in the range of 0 to 25.
2644H - 2645H	Make sure the sign bit in register A isn't set.
2646H	Load register E with the adjusted first character of the variable name in register A.
2647H - 2648H	Load register D with zero.
2649H	Save the value of the current BASIC program pointer in register pair HL on the stack.
264AH - 264CH	Load register pair HL with the starting address of the variable declaration table.
264DH	Add the value of the first character of the variable name in register pair DE to the starting address of the variable declaration table in register pair HL.
264EH	Load register D with the number type value at the location of the variable declaration table pointer in register pair HL.
264FH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
2650H	Decrement the value of the current BASIC program pointer in register pair HL.
2651H	Return.
2652H	Load register A with the value of the number type flag in register D.
2653H - 2655H	Save the number type flag for the current variable name in register A.
2656H	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
2657H - 2659H	Load register A with the FOR flag.
265AH	Test the value of the FOR flag in register A.
265BH - 265DH	Jump if a FOR statement is being processed.
265EH	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
265FH - 2660H	Check to see if the character at the location of the current BASIC program pointer in register A is a (.

2661H - 2663H	Jump if the character at the location of the current BASIC program pointer in register A is a (.
2664H	Zero register A.
2665H - 2667H	Set the array flag.
2668H	Save the value of the current BASIC program pointer in register pair HL on the stack.
2669H	Save the number type flag for the variable in register pair DE on the stack.
266AH - 266CH	Load register pair HL with the value of the simple variables pointer.
266DH	Load register pair DE with the simple variables pointer in register pair HL.
266EH - 2670H	Load register pair HL with the array variables pointer.
2671H	Go compare the value of the simple variables pointer in register pair DE with the array variables pointer in register pair HL.
2672H	Get the number type flag for the variable from stack and put it in register pair HL.
2673H - 2674H	Jump if the simple variables pointer in register pair DE is greater than or equal to the array variables pointer.
2675H	Load register A with the number type flag for the variable at the location of the simple variables pointer in register pair DE.
2676H	Load register L with the number type flag in register A.
2677H	Compare the number type flag for the variable in register H to the number type flag in register A.
2678H	Bump the value of the current simple variables pointer in register pair DE.
2679H - 267AH	Jump if the number type flags don't match.
267BH	Load register A with the first character of variable name at the location of the simple variables pointer in register pair DE.
267CH	Compare the character at the location of the simple variables pointer in register A with the first character of the variable name in register C.

267DH - 267EH	Jump if the first characters of the variable names don't match.
267FH	Bump the value of the current simple variables pointer in register pair DE.
2680H	Load register A with the second character of the variable name at the location of the simple variables pointer in register pair DE.
2681H	Compare the character at the location of the simple variables pointer in register A with the first character of the variable name in register B.
2682H - 2684H	Jump if the character at the location of the simple variables pointer in register A matches the first character of the variable name in register B.
2686H	Bump the value of the current simple variables pointer in register pair DE.
2687H	Bump the value of the simple variables pointer in register pair DE.
2688H	Save the number type flag for the variable in register pair HL on the stack.
2689H - 268AH	Load register H with zero.
268BH	Add the value of the simple variables pointer in register pair DE to the value of the number type flag in register pair HL.
268CH - 268DH	Jump.
268EH	Load register A with the number type flag for the variable in register H.
268FH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
2690H	Exchange the value of the current BASIC program pointer in register pair HL with the value on the stack.
2691H	Save the number type flag for the variable in register A on the stack.
2692H	Save the start of the array variables pointer in register pair DE on the stack.
2693H - 2695H	Load register pair DE with the return address.
2696H	Compare the return address in register pair DE with the return address in register pair HL.

2697H - 2698H	Jump if this routine is called from the VARPTR routine.
2699H - 269BH	Load register pair DE with the return address.
269CH	Compare the return address in register pair DE with the return address in register pair HL.
269DH	Get the value of the array variables pointer from the stack and put it in register pair DE.
269EH - 269FH	Jump if this routine is called to locate the variable's address.
26A0H	Get the value of the number type flag for the variable from the stack and put it in register A.
26A1H	Exchange the value of the current BASIC program pointer on the stack with the value of the return address in register pair HL.
26A2H	Save the value of the current BASIC program pointer in register pair HL on the stack.
26A3H	Save the variable's name in register pair BC on the stack.
26A4H	Load register C with the value of the number type flag for the variable in register A.
26A5H - 26A6H	Load register B with zero.
26A7H	Save the variable's number type flag in register pair BC on the stack.
26A8H	Bump the value of the variable's number type flag in register pair BC.
26A9H	Bump the value of the variable's number type flag in register pair BC.
26AAH	Bump the value of the variable's number type flag in register pair BC.
26ABH - 26ADH	Load register pair HL with the value of the free memory pointer.
26AEH	Save the value of the free memory pointer in register pair HL on the stack.
26AFH	Add the value of the variable's number type flag in register pair BC to the value of the free memory pointer in register pair HL.
26B0H	Get the value of the free memory pointer from the stack and put it in register pair BC.

26B1H	Save the value of the new free memory pointer in register pair HL on the stack.
26B2H - 26B4H	Move the array variables up in memory.
26B5H	Get the value of the new free memory pointer from the stack and put it in register pair HL.
26B6H - 26B8H	Save the value of the new free memory pointer in register pair HL.
26B9H	Load register H with the MSB of the new array variables pointer in register B.
26BAH	Load register L with the LSB of the new array variables pointer in register C.
26BBH - 26BDH	Save the value of the new array variables pointer in register pair HL.
26BEH	Decrement the value of the array variables pointer in register pair HL.
26BFH - 26C0H	Zero the location of the memory pointer in register pair HL.
26C1H	Check to see if the variable has been completely zeroed.
26C2H - 26C3H	Loop till the variable has been cleared.
26C4H	Get the value of the variable's number type flag from the stack and put it in register pair DE.
26C5H	Save the value of the number type flag in register E at the location of the memory pointer in register pair HL.
26C6H	Bump the value of the memory pointer in register pair HL.
26C7H	Get the variable's name from the stack and put it in register pair DE.
26C8H	Save the first character of the variable's name in register E at the location of the memory pointer in register pair HL.
26C9H	Bump the value of the memory pointer in register pair HL.
26CAH	Save the second character of the variable's name in register D at the location of the memory pointer in register pair HL.
26CBH	Load register pair DE with the value of the variable pointer in register pair HL.

26CCH	Bump the value of the variable pointer in register pair DE.
26CDH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
26CEH	Return.
26CFH	Load register D with the value of the current number type flag in register A.
26D0H	Load register E with the value of the current number type flag in register A.
26D1H	Clean up the stack.
26D2H	Clean up the stack.
26D3H	Exchange the value of the return address in register pair HL with the value of the current BASIC program pointer on the stack.
26D4H	Return.
26D5H - 26D7H	Zero REG1.
26D8H	Clean up the stack.
26D9H	Zero register H.
26DAH	Zero register L.
26DBH - 26DDH	Zero the string pointer location in REG1.
26DEH	Check the current value of the number type flag.
26DFH - 26E0H	Jump if the current number type isn't a string.
26E1H - 26E3H	Load register pair HL with the starting address of the Level II BASIC READY message.
26E4H - 26E6H	Save the value in register pair HL as the current string pointer.
26E7H	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
26E8H	Return.
26E9H	Save the value of the current BASIC program pointer in register pair HL on the stack.
26EAH - 26ECH	Load register pair HL with the value of the locate/create flag.
26EDH	Exchange the value of the locate/create flag in

	register pair HL with the value of the current BASIC program pointer on the stack.
26EEH	Zero register D.
26EFH	Save the variable's number type flag in register pair DE on the stack.
26F0H	Save the variable's name in register pair BC on the stack.
26F1H - 26F9H	Go evaluate the array subscript at the location of the current BASIC program pointer in register pair HL and return with the binary value in register pair DE.
26F4H	Get the variable's name from the stack and put it in register pair BC.
26F5H	Get the variable's number type flag from the stack and put it in register A.
26F6H	Exchange the value of the array subscript in register pair DE with the value of the current BASIC program pointer in register pair HL.
26F7H	Exchange the value of the array subscript in register pair HL with the value of the locate/create flag on the stack.
26F8H	Save the value of the locate/create flag in register pair HL on the stack.
26F9H	Exchange the value of the locate/create flag in register pair HL with the value of the current BASIC program pointer in register pair DE.
26F4H	Bump the number of subscripts evaluated in register A.
26FBH	Load register D with the number of subscripts evaluated in register A.
26FCH	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
26FDH - 26FEH	Check to see if the character at the location of the current BASIC program pointer in register A is a comma.
26FFH - 2700H	Jump if the character at the location of the current BASIC program pointer in register A is a comma.
2701H - 2702H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a).

2703H - 2705H	Save the value of the current BASIC program pointer in register pair HL.
2706H	Get the value of the locate/create flag from the stack and put it in register pair HL.
2707H - 2709H	Save the value of the locate/create flag in register pair HL.
270AH	Save the number of subscripts evaluated in register pair DE.
270BH - 270DH	Load register pair HL with the value of the array variables pointer.
270FH	Figure the end of the array.
2710H	Load register pair DE with the value of the array variables pointer in register pair HL.
2711H - 2713H	Load register pair HL with the value of the free memory pointer.
2714H	Exchange the value of the free memory pointer in register pair HL with the value of the array variables pointer in register pair DE.
2715H	Compare the value of the free memory pointer in register pair DE to the value of the array variables pointer in register pair HL.
2716H - 2718H	Load register A with the value of the current number type flag.
2719H - 271AH	Jump if the array variables pointer in register pair HL is greater than or equal to the value of the free memory pointer in register pair DE.
271BH	Compare the value of the variable's number type flag in register A with the value of the number type flag at the location of the array variables pointer in register pair HL.
271CH	Bump the value of the array variables pointer in register pair HL.
271DH - 271EH	Jump if the number type flags don't match.
271FH	Load register A with the first character of the variable name at the location of the array variables pointer in register pair HL.
2720H	Check to see if the first character of the variable at the location of the array variable pointer in register A matches the first character of the variable name in register C.

2721H	Bump the value of the array variables pointer in register pair HL.
2722H - 2723H	Jump if the first characters of the variable names don't match.
2724H	Load register A with the second character of the variable name at the location of the array variables pointer in register pair HL.
2725H	Compare the second character of the variable name at the location of the array variables pointer in register A matches the second character of the variable name in register B.
2727H	Bump the value of the array variables pointer in register pair HL.
2728H	Bump the value of the array variables pointer in register pair HL.
2729H	Load register E with the LSB of the offset to the next array at the location of the array variables pointer in register pair HL.
272AH	Bump the value of the array variables pointer in register pair HL.
272BH	Load register D with the MSB of the offset to the next array at the location of the array variables pointer in register pair HL.
272CH	Bump the value of the array variables pointer in register pair HL.
272DH - 272EH	Jump if the variables' names don't match.
272FH - 2731H	Load register A with the value of the locate/create flag.
2732H	Check the status of the locate/create flag in register A.
2733H - 2734H	Load register E with the DD ERROR code.
2735H - 2737H	Go to the Level II BASIC error routine and display a DD ERROR message if the locate/create flag in register A indicates the create mode.
2738H	Get the number of subscripts evaluated from the stack and put it in register A.
2739H	Compare the number of subscripts evaluated in register A with the number of subscripts for the array at the location of the array variables pointer in register pair HL.

273AH - 273CH	Jump if the number of subscripts evaluated in register A matches the number of subscripts for the array at the location of the array variables pointer in register pair HL.
273DH - 273EH	Load register E with a BS ERROR code.
273FH - 2741H	Go to the Level II BASIC error routine and display a BS ERROR message if the number of subscripts evaluated in register A doesn't match the number of subscripts for the array at the location of the array variable pointer in register pair HL.
2742H	Save the number of subscripts for the array in register A at the location of the array variables pointer in register pair HL.
2743H	Bump the value of the array variables pointer in register pair HL.
2744H	Load register E with the number type flag for the current variable.
2745H - 2746H	Zero register D.
2747H	Get the number of subscripts evaluated from the stack and put it in register A.
2748H	Save the first character of the variable's name in register C at the location of the array variables pointer in register pair HL.
2749H	Bump the value of the array variables pointer in register pair HL.
274AH	Save the second character of the variable's name in register B at the location of the array variables pointer in register pair HL.
274BH	Bump the value of the array variables pointer in register pair HL.
274CH	Load register C with the number of subscripts evaluated in register A.
274DH - 274FH	Figure the amount of memory needed.
2750H	Bump the value of the array variables pointer in register pair HL.
2751H	Bump the value of the array variables pointer in register pair HL.
2752H - 2754H	Save the value of the array variables pointer in register pair HL.
2755H	Save the number of subscripts evaluated in register

	C at the location of the array variables pointer in register pair HL.
2756H	Bump the value of the array variables pointer in register pair HL.
2757H - 2759H	Load register A with the value of the locate/create flag.
275AH	Set the flags according the status of the locate/create flag in register A.
275BH	Load register A with the number of subscripts evaluated in register C.
275CH - 275EH	Load register pair BC with the default number to be created.
275FH - 2760H	Jump if the array is being created.
2761H	Get a subscript from the stack and put it in register pair BC.
2762H	Bump the value of the subscript in register pair BC.
2763H	Save the LSB of the subscript's value in register C at the location of the array variables pointer in register pair HL.
2764H	Bump the value of the array variables pointer in register pair HL.
2765H	Save the MSB of the subscript's value in register B at the location of the array variables pointer in register pair HL.
2766H	Bump the value of the array variables pointer in register pair HL.
2767H	Save the number of subscripts evaluated in register A on the stack.
2768H - 276AH	Go multiply the size of the subscript by the value of the number type flag to determine the amount of memory necessary for the subscript.
276BH	Get the number of subscripts evaluated from the stack and put it in register A.
276CH	Check to see if there are anymore subscripts to be evaluated.
276DH - 276EH	Jump if there are anymore subscripts to be evaluated.
276FH	Save the number of subscripts to be evaluated in register A on the stack.

2770H	Load register B with the MSB of the array's length in register D.
2771H	Load register C with the LSB of the array's length in register E.
2772H	Load register pair DE with the value of the array variables pointer in register pair HL.
2773H	Add the length of the array in register pair HL to the value of the array variable pointer in register pair DE.
2774H - 2775H	Jump if overflow occurred.
2776H - 2778H	Go check to see if there is enough memory for the array.
2779H - 277BH	Get the end of the array and put it in register pair HL.
277CH	Decrement the value of the array pointer in register pair HL.
277DH - 277EH	Zero the location of the array pointer in register pair HL.
277FH	Compare the array pointer in register pair HL with the array variables pointer in register pair DE.
2780H - 2781H	Loop till the array has been cleared.
2782H	Load register pair BC with the array's length plus one.
2783H	Zero register D.
2784H - 2786H	Load register pair HL with the array variables pointer.
2787H	Load register E with the number of subscripts for the array at the location of the array variables pointer in register pair HL.
2788H	Exchange the value of the array variables pointer in register pair HL with the number of subscripts for the array in register pair DE.
2789H	Multiply the number of subscripts for the array in register pair HL by two.
278AH	Add the length of the array in register pair BC to the number of subscripts times two in register pair HL.
278BH	Exchange the array's length in register pair HL with the array variables pointer in register pair DE.

278CH	Decrement the value of the array variables pointer in register pair HL.
278DH	Decrement the value of the array variables pointer in register pair HL.
278EH	Save the LSB of the offset to the next array in register E at the location of the array variables pointer in register pair HL.
278FH	Bump the value of the array variables pointer in register pair HL.
2790H	Save the MSB of the offset to the next array in register D at the location of the array variables pointer in register pair HL.
2791H	Bump the value of the array variables pointer in register pair HL.
2792H	Get the value of the locate/create flag from the stack and put it in register A.
2793H - 2794H	Jump if the array is being created.
2795H	Zero register B.
2796H	Zero register C.
2797H	Load register A with the number of subscripts for the array at the location of the array variables pointer in register pair HL.
2798H	Bump the value of the array variables pointer in register pair HL.
279AH	Get the array variables pointer from the stack and put it in register pair HL.
279BH	Load register E with the LSB of the subscript limit at the location of the array variables pointer in register pair HL.
279CH	Bump the value of the array variables pointer in register pair HL.
279DH	Load register D with the MSB of the subscript limit at the location of the array variables pointer in register pair HL.
279EH	Bump the value of the array variables pointer in register pair HL.
279FH	Exchange the value of the array variables pointer in register pair HL with the subscript on the stack.
27A0H	Save the number of subscripts for the array in register A on the stack.

27A1H	Compare the subscript limit in register pair DE with the subscript for the array in register pair HL.
27A2H - 27A4H	Jump if the subscript for the array in register pair HL is greater than the subscript limit in register pair DE.
27A5H - 27A7H	Multiply the previous subscript by the subscript limit.
27A8H	Add the subscript limit for the array in register pair DE to the array pointer in register pair HL.
27A9H	Get the number of subscripts for the array from the stack and put it in register A.
27AAH	Check to see if there are anymore subscripts to be evaluated.
27ABH	Load register B with the MSB of the subscript pointer in register H.
27ACH	Load register C with the LSB of the subscript pointer in register L.
27ADH - 27AEH	Loop till all of the subscripts have been evaluated.
27AFH - 27B1H	Get the number type flag for the current array.
27B2H	Load register B with the MSB of the subscript pointer in register H.
27B3H	Load register C with the LSB of the subscript pointer in register L.
27B4H	Multiply the subscript pointer in register pair HL by two.
27B5H - 27B6H	Check the value of the number type flag in register A.
27B7H - 27B8H	Jump if the current number type is an integer or string.
27B9H	Multiply the subscript pointer in register pair HL by two.
27BAH - 27BBH	Jump if the current number type is single precision.
27BCH	Multiply the subscript pointer in register pair HL by two.
27BDH	Set the flags.
27BEH - 27C0H	Jump if the current number type isn't a string.
27C1H	Add the value of the original subscript pointer in

	register pair BC to the subscript pointer in register pair HL.
27C2H	Get the value of the array variables pointer from the stack and put it in register pair BC.
27C3H	Add the value of the array variables pointer in register pair BC to the subscript pointer in register pair HL.
27C4H	Load register pair DE with the variable pointer in register pair HL.
27C5H - 27C7H	Get the value of the current BASIC program pointer and put it in register pair HL.
27C8H	Return.
27C9H - 27D3H	**LEVEL II BASIC MEM ROUTINE**
27C9H	Zero register A.
27CAH	Save the value of the current BASIC program pointer in register pair HL on the stack.
27CBH - 27CDH	Zero the number type flag.
27CEH - 27D0H	Call the Level II BASIC FRE routine.
27D1H	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
27D2H	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
27D3H	Return.
27D4H - 27F4H	**LEVEL II BASIC FRE ROUTINE**
27D4H - 27D6H	Load register pair HL with the start of free memory pointer.
27D7H	Load register pair DE with the value of the free memory pointer in register pair HL.
27D8H - 27DAH	Zero register pair HL.
27DBH	Add the value in register pair HL to the current value of the stack pointer.
27DCH	Check the value of the current number type flag.
27DDH - 27DEH	Jump if this routine is called from the MEM routine.
27DFH - 27E1H	Go update the string pointers.

27E2H - 27E4H Go check to see how much string space remains.

27E5H - 27E7H Load register pair HL with the start of string space pointer.

27E8H Load register pair DE with the start of string space pointer in register pair HL.

27E9H - 27EBH Load register pair HL with the next available location in string space pointer.

27ECH Load register A with the LSB of the next available location in string space pointer in register L.

27EDH Subtract the LSB of the start of string space pointer in register E from the LSB of the next available location in string space pointer in register A.

27EEH Load register L with the LSB of the amount of string space remaining in register A.

27EFH Load register A with the MSB of the next available location in string space pointer in register H.

27F0H Subtract the MSB of the string space pointer in register D from the MSB of the next available location of string space pointer in register A.

27F1H Load register H with the MSB of the amount of string space remaining in register A.

27F2H - 27F4H Jump.

27F5H - 27FDH LEVEL II BASIC POS ROUTINE

27F5H - 27F7H Load register A with the current cursor line position.

27F8H Load register L with the value of the current cursor line position in register A.

27F9H Zero register A.

27FAH Load register H with zero.

27FBH - 27FDH Jump.

27FEH - 2818H LEVEL II BASIC USR ROUTINE

27FEH - 2800H Go call the DOS link at 41A9H.

2801H Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.

2802H - 2804H Go evaluate the expression at the location of the

	current BASIC program pointer in register pair HL and return with the result in REG1.
2805H	Save the value of the current BASIC program pointer in register pair HL on the stack.
2806H - 2808H	Load register pair HL with the return address.
2809H	Save the value of the return address in register pair HL on the stack.
280AH - 280CH	Load register A with the value of the current number type flag.
280DH	Save the value of the current number type flag in register A.
280EH - 280FH	Check to see if the current value in REG1 is a string.
2810H - 2812H	If the current result in REG1 is a string then go get the string address in register pair HL.
2813H	Get the value of the current number type flag from the stack and put it in register A.
2814H	Load register pair DE with the value of the string address in register pair HL.
2815H - 2817H	Load register pair HL with the starting address of the machine language subroutine.
2818H	Jump to the address in register pair HL.
2819H - 2827H	**CONVERSION ROUTINE**
2819H	Save the value in register pair HL on the stack.
281AH - 281BH	Mask the value of the current number type flag in register A.
281CH - 281EH	Load register pair HL with the address of the arithmetic conversion routines.
281FH	Load register C with the value of the number type flag in register A.
2820H - 2821H	Zero register B.
2823H - 2825H	Go convert the current result in REG1 to it's proper number type.
2826H	Get the value from the stack and put it in register pair HL.
2827H	Return.
2828H - 2835H	**INPUT ROUTINE**

2828H	Save the value of the current BASIC program pointer in register pair HL on the stack.
2829H - 282BH	Load register pair HL with the value of the current BASIC line number.
282CH	Bump the value of the current BASIC line number in register pair HL.
282DH	Load register A with the MSB of the current BASIC line number in register H.
282EH	Combine the LSB of the current BASIC line number in register L with the MSB of the current BASIC line number in register A.
282FH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
2830H	Return if this isn't the command mode.
2831H - 2832H	Load register E with the ID ERROR code.
2833H - 2835H	Go to the Level II BASIC error routine and display an ID ERROR message if this is the command mode.
2836H - 2856H	**STRING ROUTINE**
2836H - 2838H	Go convert the current result in REG1 to an ASCII string.
2839H - 283BH	Go make a temporary string work area entry.
283CH - 283EH	Load register pair HL with the string's VARPTR.
283FH - 2841H	Load register pair BC with the return address.
2842H	Save the value of the return address in register pair BC on the stack.
2843H	Load register A with the string's length at the location of the string's VARPTR in register pair HL.
2844H	Bump the value of the string's VARPTR in register pair HL.
2845H	Save the value of the string's VARPTR in register pair HL on the stack.
2846H - 2848H	Go make sure there is enough string space for the string.
2849H	Get the value of the string's VARPTR from the stack and put it in register pair HL.
284AH	Load register C with the LSB of the string's address

	at the location of the string's VARPTR in register pair HL.
284BH	Bump the value of the string's VARPTR in register pair HL.
284CH	Load register B with the MSB of the string's address at the location of the string's VARPTR in register pair HL.
284DH - 284FH	Go save the string's length and the string's address.
2850H	Save the value in register pair HL on the stack.
2851H	Load register L with the value of the string's length in register A.
2852H - 2854H	Go move thè string from it's temporary location to string space.
2855H	Get the value from the stack and put it in register pair DE.
2856H	Return.
2857H - 2864H	**STRING ROUTINE**
2857H - 2859H	Go make sure that there is enough string space remaining for the string.
285AH - 285CH	Load register pair HL with the address of the temporary string parameter storage area.
285DH	Save the address of the temporary string parameter area in register pair HL on the stack.
285EH	Save the string's length in register A at the location of the temporary string parameter storage pointer in register pair HL.
285FH	Bump the value of the temporary string parameter storage pointer in register pair HL.
2860H	Save the LSB of the string's address in register E at the location of the temporary string parameter storage pointer in register pair HL.
2861H	Bump the value of the temporary string parameter storage pointer in register pair HL.
2862H	Save the MSB of the string's address in register D at the location of the temporary string parameter storage pointer in register pair HL.
2863H	Get the address of the temporary string parameter storage area from the stack and put it in register pair HL.

2864H	Return.
2865H - 28A5H	**STRING ROUTINE**
2865H	Decrement the value of the current BASIC program pointer in register pair HL.
2866H - 2867H	Load register B with a quote.
2868H	Load register D with a quote.
2869H	Save the address of the current BASIC program pointer in register pair HL on the stack.
286AH - 286BH	Load register C with a −1.
286CH	Bump the value of the current BASIC program pointer in register pair HL.
286DH	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
286EH	Bump the value of the counter in register C.
286FH	Check to see if the character at the location of the current BASIC program pointer in register A is an end of the BASIC line character.
2870H - 2871H	Jump if the character at the location of the current BASIC program pointer in register A is an end of the BASIC line character.
2872H	Check to see if the character at the location of the current BASIC program pointer in register A is the same as the terminating character in register D.
2873H - 2874H	Jump if the character at the location of the current BASIC program pointer in register A is the same as the terminating character in register D.
2875H	Check to see if the character at the location of the current BASIC program pointer in register A is the same at the terminating character in register B.
2876H - 2877H	Loop if the character at the location of the current BASIC program pointer in register A isn't the same as the terminating character in register B.
2878H - 2879H	Check to see if the character at the location of the current BASIC program pointer in register A is a quote.
287AH - 287CH	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character if the character at the location of the cur-

	rent BASIC program pointer in register A was a quote.
287DH	Exchange the value of the current BASIC program pointer in register pair HL with the string's address on the stack.
287EH	Bump the string's address in register pair HL till it points to the first character of the string.
287FH	Load register pair DE with the string's address in register pair HL.
2880H	Load register A with the string's length in register C.
2881H - 2883H	Go save the string's length and the string's address.
2884H - 2886H	Load register pair DE with the address of the string parameter storage area.
2889H - 288BH	Load register pair HL with the next available location in the temporary string work area in register pair HL as the string's VARPTR in REG1.
288CH - 288EH	Save the value of the next available location in the temporary string work area in register pair HL as the string's VARPTR in REG1.
288FH - 2890H	Load register A with the string number type flag.
2891H - 2893H	Save the value in register A as the current number type flag.
2894H - 2896H	Add the length of the string to the next available location in the temporary string work area in register pair HL.
2897H - 2899H	Load register pair DE with the ending address of the temporary string work area.
289AH	Check to see if the updated temporary string work area location in register pair HL isn't greater than the ending address of the temporary string work area in register pair DE.
289BH - 289DH	Save the updated string work area location in register pair HL as the next available location in the temporary string work area.
289EH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
289FH	Load register A with the character at the location of the current BASIC program pointer in register pair HL.

28A0H	Return if the updated temporary string work area location wasn't beyond the end of the temporary string work area.
28A1H - 28A2H	Load register E with a ST ERROR code.
28A3H - 28A5H	Go to the Level II BASIC error routine and display a ST ERROR message if the temporary string work area has overflowed.
28A6H - 28BEH	**DISPLAY MESSAGE ROUTINE**
28A6H	Bump the value of the current BASIC program pointer in register pair HL.
28A7H - 28A9H	Go build a temporary string work area entry for the message at the location of the current BASIC program pointer in register pair HL.
28AAH - 28ACH	Load register pair HL with the string's VARPTR.
28ADH - 28AFH	Go get the string's length in register D and the string's address in register pair BC.
28B0H	Bump the value of the string's length in register D.
28B1H	Decrement the value of the string's length in register D.
28B2H	Return if all of the characters in the string have been sent to the current output device.
28B3H	Load register A with the character at the location of the string pointer in register pair BC.
28B4H - 28B6H	Go send the character in register A to the current output device.
28B7H - 28B8H	Check to see if the character in register A is a carriage return.
28B9H - 28BBH	Jump if the character in register A is a carriage return.
28BCH	Bump the value of the string pointer in register pair BC.
28BDH - 28BEH	Loop till all of the characters in the string have been sent to the current output device.
28BFH - 28D9H	**STRING ROUTINE**
28BFH	Set the flags.
28C1H	Get the string's length from the stack and put it in register A.

28C2H	Save the length of the string in register A on the stack.
28C3H - 28C5H	Load register pair HL with the start of the string space pointer.
28C6H	Load register pair DE with the start of the string space pointer in register pair HL.
28C7H - 28C9H	Load register pair HL with the next available location in string space pointer.
28CAH	Complement the string's length in register A.
28CBH	Load register C with the negative string's length in register A.
28CCH - 28CDH	Load register B with a −1.
28CEH	Add the negative string's length in register pair BC to the next available location in string space pointer in register pair HL.
28CFH	Bump the value of the adjusted next available location in string space pointer in register pair HL.
28D0H	Check to see if the adjusted next available location in string space pointer in register pair HL is less than the start of string space pointer in register pair DE.
28D1H - 28D2H	Jump if the adjusted next available location in string space pointer in register pair HL is less than the start of string space pointer in register pair DE.
28D3H - 28D5H	Save the value in register pair HL as the next available location in string space pointer.
28D6H	Bump the value of the next available location in string space pointer in register pair HL.
28D7H	Load register pair DE with the next available location in string space pointer in register pair HL.
28D8H	Get the string's length from the stack and put it in register A.
28D9H	Return.
28DAH - 298EH	**STRING ROUTINE**
28DAH	Get the length of the string from the stack and put it in register A.
28DBH - 28DCH	Load register E with an OS ERROR code.
28DDH - 28DFH	Go to the Level II BASIC error routine and display

	an OS ERROR message if there isn't enough string space available for the string.
28E0H	Set the flags.
28E1H	Save the string's length in register A on the stack.
28E2H - 28E4H	Load register pair BC with the return address.
28E5H	Save the value of the return address in register pair BC on the stack.
28E6H - 28E8H	Load register pair HL with the top of memory pointer.
28E9H - 28EBH	Save the top of BASIC memory pointer in register pair HL as the next available location in string space pointer.
28ECH - 28EEH	Zero register pair HL.
28EFH	Save the value in register pair HL on the stack.
28F0H - 28F2H	Load register pair HL with the start of string space pointer.
28F3H	Save the start of string space pointer in register pair HL on the stack.
28F4H - 28F6H	Load register pair HL with the start of the temporary string work area pointer.
28F7H	Load register pair DE with the start of the temporary string work area pointer in register pair HL.
28F8H - 28FAH	Load register pair HL with the next available location in the temporary string work area pointer.
28FBH	Exchange the start of the temporary string work area pointer in register pair DE with the next available location in the temporary string work area pointer in register pair HL.
28FCH	Check to see if the start of the temporary string work area pointer in register pair HL is the same as the next available location in the temporary string work area pointer.
28FDH - 28FFH	Load register pair BC with the return address.
2900H - 2902H	Jump if the temporary string work area isn't empty.
2903H	Load register pair HL with the start of the simple variables pointer.
2906H	Load register pair DE with the simple variables pointer in register pair HL.

2907H - 2909H	Load register pair HL with the start of the array variables pointer.
290AH	Exchange the value of the simple variables pointer in register pair DE with the value of the array variables pointer in register pair HL.
290BH	Check to see if the simple variables pointer in register pair HL is the same as the array variables pointer in register pair DE.
290CH - 290DH	Jump if the simple variables pointer in register pair HL is the same as the array variables pointer in register pair DE.
290EH	Load register A with the number type flag at the location of the simple variables pointer in register pair HL.
290FH	Bump the value of the simple variables pointer in register pair HL.
2910H	Bump the simple variables pointer in register pair HL.
2911H	Bump the value of the simple variables pointer in register pair HL.
2912H - 2913H	Check to see if the variable at the location of the simple variables pointer is a string.
2914H - 2915H	Jump if the variable at the location of the simple variables pointer in register pair HL isn't a string.
2916H - 2918H	Go do string check.
2919H	Zero register A.
291AH	Zero register E.
291BH - 291CH	Zero register D.
291DH	Add the value of the number type flag in register pair DE to the simple variables pointer in register pair HL.
291EH - 291FH	Loop till all of the simple variables have been checked.
2920H	Clean up the stack.
2921H	Load register pair DE with the value of the array variables pointer in register pair HL.
2922H - 2924H	Load register pair HL with the start of free memory pointer.

2925H	Exchange the value of the array variables pointer in register pair DE with the value of the free memory pointer in register pair HL.
2926H	Check to see if the array variables pointer in register pair HL is the same as the free memory pointer in register pair DE.
2927H - 2929H	Jump if the array variables pointer in register HL is the same as the start of the free memory pointer in register pair DE.
292AH	Load register A with the number type flag at the location of the array variables pointer in register pair HL.
292BH	Bump the value of the array variables pointer in register pair HL.
292CH - 292EH	Go get the offset to the next array and return with it in register pair BC.
292FH	Save the value of the array variables pointer in register pair HL on the stack.
2930H	Add the value of the offset to the next array in register pair BC to the value of the array variables pointer in register pair HL.
2931H - 2932H	Check to see if the array being examined is a string.
2933H - 2934H	Jump if the array being examined isn't a string.
2935H - 2937H	Save the address of the next array in register pair HL.
2938H	Get the value of the array variables pointer from the stack and put it in register pair HL.
2939H	Load register C with the number of subscripts for the array at the location of the array variables pointer in register pair HL.
293AH - 293BH	Zero register B.
293CH	Add the number of subscripts in the array in register pair BC to the value of the array variables pointer in register pair HL.
293DH	Add the number of subscripts in the array in register pair BC to the value of the array variables pointer in register pair HL.
293EH	Bump the value of the array variables pointer in register pair HL.

293FH	Load register pair DE with the value of the array variables pointer in register pair HL.
2940H - 2942H	Load register pair HL with the address of the next array.
2943H	Exchange the value of the array variables pointer in register pair DE with the address of the array in register pair HL.
2944H	Check to see if the array variables pointer in register pair HL is the same as the address of the next array in register pair DE.
2945H - 2946H	Jump if the array variables pointer in register pair HL is the same as the address of the next array in register pair DE.
2947H - 2948H	Load register pair BC with the return address.
294AH	Save the value in register pair BC on the stack.
294BH	Zero register A.
294CH	Load register A with the length of the string at the location of the array variables pointer in register pair HL.
294DH	Bump the value of the array variables pointer in register pair HL.
294EH	Load register E with the LSB of the string's address at the location of the array variables pointer in register pair HL.
294FH	Bump the value of the array variables pointer in register pair HL.
2950H	Load register D with the MSB of the string's address at the location of the array variables pointer in register pair HL.
2951H	Bump the value of the array variables pointer in register pair HL.
2952H	Return if the string's length in register A is equal to zero.
2953H	Load register B with the MSB of the array variables pointer in register H.
2954H	Load register C with the LSB of the array variables pointer in register L.
2955H - 2957H	Load register pair HL with the location of the next available location in string space pointer.

2958H	Check to see if the string's address in register pair DE is in string space.
2959H	Load register H with the MSB of the array variables pointer in register B.
295AH	Load register L with the LSB of the array variables pointer in register C.
295BH	Return if the string's address in register pair DE is in string space.
295CH	Get the return address from the stack and put it in register pair HL.
295DH	Exchange the value of the return address in register pair HL with the start of string space pointer on the stack.
295EH	Check to see if the string's address in register pair DE is below the start of string space pointer in register pair HL.
295FH	Exchange the start of string space pointer in register pair HL with the value of the return address on the stack.
2960H	Save the value of the return address in register pair HL on the stack.
2961H	Load register H with the MSB of the array variables pointer in register B.
2962H	Load register L with the LSB of the array variables pointer in register C.
2963H	Return if the string's address in register pair DE is below the string space pointer.
2964H	Get the return address from the stack and put it in register pair BC.
2965H	Clean up the stack.
2966H	Clean up the stack.
2967H	Save the value of the array variables pointer in register pair HL on the stack.
2968H	Save the string's address in register pair DE on the stack.
2969H	Save the value of the return address in register pair BC on the stack.
296AH	Return.

296BH	Load register pair DE with the address of the last string put into the temporary string work area.
296CH	Get the temporary string work area pointer from the stack and put it in register pair HL.
296DH	Load register A with the LSB of the temporary string work area pointer in register L.
296EH	Combine the MSB of the temporary string work area pointer in register H with the LSB of the temporary string work area pointer in register A.
296FH	Return if the temporary string work area is empty.
2970H	Decrement the value of the temporary string work area pointer in register pair HL.
2971H	Load register B with the MSB of the string's address at the location of the temporary string work area pointer in register pair HL.
2972H	Decrement the value of the temporary string work area pointer in register pair HL.
2973H	Load register C with the LSB of the string's address at the location of the temporary string work area pointer in register pair HL.
2974H	Save the value of the temporary string work area pointer in register pair HL on the stack.
2975H	Decrement the value of the temporary string work area pointer in register pair HL.
2976H	Load register L with the string's length at the location of the temporary string work area pointer in register pair HL.
2977H - 2978H	Zero register H.
2979H	Add the length of the string in register pair HL to the string's address in register pair BC.
297AH	Load register D with the MSB of the string's address in register B.
297BH	Load register E with the LSB of the string's address in register C.
297CH	Decrement the value of the string's ending address in register pair HL.
297DH	Load register B with the MSB of the string's ending address in register H.

297EH	Load register C with the LSB of the string's ending address in register L.
297FH - 2981H	Load register pair HL with the next available location in string space pointer.
2982H - 2984H	Move the string from the temporary storage location to string space.
2985H	Get the temporary string work area pointer from the stack and put it in register pair HL.
2986H	Save the LSB of the string's address in register C at the location of the temporary string work area pointer in register pair HL.
2987H	Bump the value of the temporary string work area pointer in register pair HL.
2988H	Save the MSB of the string's address in register B at the location of the temporary string work area pointer in register pair HL.
2989H	Load register L with the LSB of the string's address in register C.
298AH	Load register H with the MSB of the string's address in register B.
298BH	Decrement the string's address in register pair HL.
298CH - 298EH	Jump.
298FH - 29C5H	**STRING ADDITION ROUTINE**
298FH	Save the operator value in register pair BC on the stack.
2990H	Save the value of the current BASIC program pointer in register pair HL on the stack.
2991H - 2993H	Load register pair HL with the first string's VARPTR in REG1.
2994H	Exchange the value of the first string's VARPTR in register pair HL with the value of the current BASIC program on the stack.
2995H - 2997H	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL.
2998H	Exchange the value of the current BASIC program pointer in register pair HL with the value of the first string's VARPTR on the stack.
2999H - 299BH	Go make sure the current result in REG1 is a string.

299CH	Load register A with the first string's length at the location of the first string's VARPTR in register pair HL.
299DH	Save the value of the first string's VARPTR in register pair HL on the stack.
299EH - 29A0H	Load register pair HL with the second string's VARPTR in REG1.
29A1H	Save the second string's VARPTR in register pair HL on the stack.
29A2H	Add the length of the second string at the location of the second string's VARPTR in register pair HL to the length of the first string in register A.
29A3H - 29A4H	Load register E with a LS ERROR code.
29A5H - 29A7H	Go to the Level II BASIC error routine and display a LS ERROR message if the combined lengths of the strings is greater than 255.
29A8H - 29AAH	Go make sure that there is enough string space for the new string.
29ABH	Get the second string's VARPTR from the stack and put it in register pair DE.
29ACH - 29AEH	Go update the temporary string work area.
29AFH	Exchange the second string's VARPTR in register pair HL with the first string's VARPTR on the stack.
29B0H - 29B2H	Go update the temporary string work area.
29B3H	Save the value of the first string's VARPTR in register pair HL on the stack.
29B4H - 29B6H	Load register pair HL with the second string's address.
29B7H	Load register pair DE with the second string's address in register pair HL.
29B8H - 29BAH	Go move the first string to it's temporary storage location.
29BBH - 29BDH	Go move the second string to it's temporary storage location.
29BEH - 29C0H	Load register pair HL with the return address.
29C1H	Exchange the value of the return address in register pair HL with the value of the current BASIC program pointer on the stack.

29C2H	Save the value of the current BASIC program pointer in register pair HL on the stack.
29C3H - 29C5H	Jump.
29C6H - 29D6H	**STRING ROUTINE**
29C6H	Load register pair HL with the value of the return address on the stack.
29C7H	Exchange the value of the return address in register pair HL with the string's VARPTR on the stack.
29C8H	Load register A with the string's length at the location of the string's VARPTR in register pair HL.
29C9H	Bump the value of the string's VARPTR in register pair HL.
29CAH	Load register C with the LSB of the string's address at the location of the string's VARPTR in register pair HL.
29CBH	Bump the value of the string's VARPTR in register pair HL.
29CCH	Load register B with the MSB of the string's address at the location of the string's VARPTR in register pair HL.
29CDH	Load register L with the string's length in register A.
29CEH	Bump the value of the string's length in register L.
29CFH	Decrement the value of the string's length in register L.
29D0H	Return if all of the characters in the string have been moved.
29D1H	Load register A with the character at the location of the string pointer in register pair BC.
29D2H	Save the character in register A at the location of the string storage pointer in register pair DE.
29D3H	Bump the value of the string pointer in register pair BC.
29D4H	Bump the value of the string storage pointer in register pair DE.
29D5H - 29D6H	Loop till the string has been completely moved.
29D7H - 29F4H	**STRING ROUTINE**

29D7H - 29D9H	Go make sure that the current result in REG1 is a string.
29DAH - 29DCH	Load register pair HL with the string's VARPTR in REG1.
29DDH	Load register pair DE with the value of the string's VARPTR in register pair HL.
29DEH - 29E0H	Check to see if the string is the last entry in the temporary string work area.
29E1H	Load register pair HL with the value of the string's VARPTR in register pair DE.
29E2H	Return if the string isn't the last entry in the temporary string work area.
29E3H	Save the value of the string's VARPTR in register pair DE on the stack.
29E4H	Load register D with the MSB of the string's address in register B.
29E5H	Load register E with the LSB of the string's address in register C.
29E6H	Decrement the value of the string's address in register pair DE.
29E7H	Load register C with the string's length at the location of the string's VARPTR in register pair HL.
29E8H - 29EAH	Load register pair HL with the next available location in string space pointer.
29EBH	Check to see if the string's address in register pair DE is the same as the next available location in string space pointer in register pair HL.
29ECH - 29EDH	Jump if the string's address in register pair DE isn't the same as the next available location in string space pointer in register pair HL.
29EEH	Load register B with the value in register A.
29EFH	Add the length of the string in register pair BC to the next available location in string space pointer in register pair HL.
29E0H - 29F2H	Save the adjusted next available location in string space pointer in register pair HL.
29F3H	Get the string's VARPTR from the stack and put it in register pair HL.
29F4H	Return.

29F5H - 2A02H STRING ROUTINE

29F5H - 29F7H Load register pair HL with the temporary string work area pointer.

29F8H Decrement the value of the temporary string work area pointer in register pair HL.

29F9H Load register B with the MSB of the string's address at the location of the temporary string work area pointer in register pair HL.

29FAH Decrement the value of the temporary string work area pointer in register pair HL.

29FBH Load register C with the LSB of the string's address at the location of the temporary string work area pointer in register pair HL.

29FCH Decrement the value of the temporary string work area pointer in register pair HL.

29FDH Check to see if the string's VARPTR in register pair DE matches the temporary string work area pointer in register pair HL.

29FEH Return if the string's VARPTR in register pair DE doesn't match the temporary string work area pointer in register pair HL.

29FFH - 2A01H Save the value of the temporary string work area pointer in register pair HL.

2A02H Return.

2A03H - 2A0EH LEVEL II BASIC LEN ROUTINE

2A03H - 2A05H Load register pair BC with the return address.

2A06H Save the value of the return address in register pair BC on the stack.

2A07H - 2A09H Go get the string's VARPTR and put it in register pair HL.

2A0AH Zero register A.

2A0BH Zero register D.

2A0CH Load register A with the string's length at the location of the string's VARPTR in register pair HL.

2A0DH Set the flags according to the string's length in register A.

2A0EH Return.

2A0FH - 2A1EH	**LEVEL II BASIC ASC ROUTINE**
2A0FH - 2A11H	Load register pair BC with the return address.
2A12H	Save the value of the return address in register pair BC on the stack.
2A13H - 2A15H	Go get string's VARPTR into register pair HL and the string's length into register A.
2A16H - 2A18H	Go to the Level II BASIC error routine and display a FC ERROR message if the string's length in register A is equal to zero.
2A19H	Bump the value of the string's VARPTR in register pair HL.
2A1AH	Load register E with the LSB of the string's address at the location of the string's VARPTR in register pair HL.
2A1BH	Bump the value of the string's VARPTR in register pair HL.
2A1CH	Load register D with the MSB of the string's address at the location of the string's VARPTR in register pair HL.
2A1DH	Load register A with the character at the location of the string pointer in register pair DE.
2A1EH	Return.
2A1FH - 2A2EH	**LEVEL II BASIC CHR$ ROUTINE**
2A1FH - 2A20H	Load register A with the string's length.
2A21H - 2A23H	Go save the string's length in register A and set up the string's address.
2A24H - 2A26H	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the integer result in register pair DE.
2A27H - 2A29H	Load register pair HL with the string's address.
2A2AH	Save the character in register E at the location of the string pointer in register pair HL.
2A2BH	Clean up the stack.
2A2CH - 2A2EH	Jump.
2A2FH - 2A60H	**LEVEL II BASIC STRING$ ROUTINE**
2A2FH	Go bump the value of the current BASIC program

	pointer in register HL till it points to the next character.
2A30H - 2A31H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a (.
2A32H - 2A34H	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the integer result in register pair DE.
2A35H	Save the string's length in register pair DE on the stack.
2A36H - 2A37H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a comma.
2A38H - 2A3AH	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the result in REG1.
2A3BH - 2A3CH	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a).
2A3DH	Exchange the value of the current BASIC program pointer in register pair HL with the string's length on the stack.
2A3EH	Save the string's length in register pair HL on the stack.
2A3FH	Go check the current value of the number type flag.
2A40H - 2A41H	Jump if the current result in REG1 is a string.
2A42H - 2A44H	Convert the current result in REG1 to an integer and return with the 8-bit result in register A.
2A45H - 2A46H	Jump.
2A47H - 2A49H	Go get the first character in the string and return with it in register A.
2A4AH	Get the string's length from the stack and put it in register pair DE.
2A4BH	Save the character for the string in register A on the stack.
2A4CH	Save the character for the string in register A on the stack.
2A4DH	Load register A with the string's length in register E.

2A4EH - 2A4FH	Go update the temporary string work area.
2A51H	Load register E with the string's length in register A.
2A52H	Get the character for the string from the stack and put it in register A.
2A53H	Bump the value of the string's length in register E.
2A54H	Decrement the string's length in register E.
2A55H - 2A56H	Jump if the string has been completed.
2A57H - 2A59H	Load register pair HL with the string's address.
2A5AH	Save the character in register A at the location of the string pointer in register pair HL.
2A5BH	Bump the value of the string pointer in register pair HL.
2A5CH	Decrement the string's length in register E.
2A5DH - 2A5EH	Loop till the string has been completed.
2A5FH - 2A60H	Jump.
2A61H - 2A90H	**LEVEL II BASIC C LEFT$ ROUTINE**
2A61H - 2A63H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a).
2A64H	Zero register A.
2A65H	Exchange the value of the current BASIC program pointer in register pair HL with the string's VARPTR on the stack.
2A66H	Zero register C.
2A68H	Load register H with the value in register A.
2A69H	Save the value of the string's VARPTR in register pair HL on the stack.
2A6AH	Load register A with the string's length at the location of the string's VARPTR in register pair HL.
2A6BH	Check to see if the new string's length in register B is greater than the string's length in register A.
2A6CH - 2A6DH	Jump if the new string's length in register B is greater than the string's length in register A.
2A6EH	Load register A with the new string's length in register B.

2A70H - 2A71H	Zero register C.
2A72H	Save the string's length in register pair BC on the stack.
2A73H - 2A75H	Go see if there is enough string space for the new string.
2A76H	Get the new string's length from the stack and put it in register pair BC.
2A77H	Get the string's VARPTR from the stack and put it in register pair HL.
2A78H	Save the string's VARPTR in register pair HL on the stack.
2A79H	Bump the value of the string's VARPTR in register pair HL.
2A7AH	Load register B with the LSB of the string's address at the location of the string's VARPTR in register pair HL.
2A7BH	Bump the value of the string's VARPTR in register pair HL.
2A7CH	Load register H with the MSB of the string's address at the location of the string's VARPTR in register pair HL.
2A7DH	Load register L with the LSB of the string's address in register B.
2A7EH - 2A7FH	Zero register B.
2A80H	Add the string's length in register pair BC to the string's address in register pair HL.
2A81H	Load register B with the MSB of the string's ending address in register H.
2A82H	Load register C with the LSB of the string's ending address in register L.
2A83H - 2A85H	Go save the string's length and the string's address.
2A86H	Load register L with the string's length in register A.
2A87H - 2A89H	Go move the string into string space.
2A8AH	Clean up the stack.
2A8BH - 2A8DH	Go update the temporary string work area.
2A8EH - 2A90H	Jump.

2A91H - 2A99H LEVEL II BASIC RIGHT$ ROUTINE

2A91H - 2A93H Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a).

2A94H Get the string's VARPTR from the stack and put it in register pair DE.

2A95H Save the string's VARPTR in register pair DE on the stack.

2A96H Load register A with the string's length at the location of the string's VARPTR in register pair DE.

2A97H Subtract the new string's length in register B from the string's length in register A.

2A98H - 2A99H Jump.

2A9AH - 2AC4H LEVEL II BASIC MID$ ROUTINE

2A9AH Load register pair HL with the value of the current BASIC program pointer in register pair DE.

2A9BH Load register A with the character at the location of the current BASIC program pointer in register pair HL.

2A9CH - 2A9EH Go get the string's position in register B and the string's VARPTR in register pair DE.

2A9FH Bump the value of the string's position in register B.

2AA0H Decrement the value of the string's position in register B.

2AA1H - 2AA3H Go to the Level II BASIC error routine and display a FC ERROR message if the string's position in register B is equal to zero.

2AA4H Save the value of the string's position in register B on the stack.

2AA5H - 2AA6H Load register E with the default string's length.

2AA7H - 2AA8H Check to see if the character at the location of the current BASIC program pointer in register A is a).

2AA9H - 2AAAH Jump if the character at the location of the current BASIC program pointer in register A is a).

2AABH - 2AACH Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a comma.

2AADH - 2AAFH Go evaluate the expression at the location of the

	current BASIC program pointer in register pair HL and return with the integer result in register pair DE.
2AB0H - 2AB1H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a).
2AB2H	Get the value of the string's position from the stack and put it in register A.
2AB3H	Exchange the value of the current BASIC program pointer in register pair HL with the value of the string's VARPTR on the stack.
2AB4H - 2AB6H	Load register pair BC with the return address.
2AB7H	Save the return address in register pair BC on the stack.
2AB8H	Decrement the value of the string's position in register A.
2AB9H	Compare the string's length at the location of the string's VARPTR in register pair HL with the value of the string's position in register A.
2ABAH - 2ABBH	Zero register B.
2ABCH	Return if the string's position in register A is greater than the string's length at the location of the string's VARPTR in register pair HL.
2ABDH	Load register C with the string's position in register A.
2ABEH	Load register A with the string's length at the location of the string's VARPTR in register pair HL.
2ABFH	Subtract the value of the string's position in register C from the string's length in register A.
2AC0H	Compare the new string's length in register E with the adjusted string's length in register A.
2AC1H	Load register B with the adjusted string's length in register A.
2AC2H	Return if the new string's length in register E is greater than the string's length in register A.
2AC3H	Load register B with the new string's length in register E.
2AC4H	Return.

2AC5H - 2ADEH LEVEL II BASIC VAL ROUTINE

2AC5H - 2AC7H	Go get the string's length in register A and the string's VARPTR in register pair HL.
2AC8H - 2ACAH	Jump if the string's length in register A is equal to zero.
2ACBH	Load register E with the string's length at the location of the string's VARPTR in register pair HL.
2ACCH	Bump the value of the string's VARPTR in register pair HL.
2ACDH	Load register A with the LSB of the string's address at the location of the string's VARPTR in register pair HL.
2ACEH	Bump the value of the string's VARPTR in register pair HL.
2ACFH	Load register H with the MSB of the string's address at the location of the string's VARPTR in register pair HL.
2AD0H	Load register L with the LSB of the string's address in register A.
2AD1H	Save the value of the string's address in register pair HL on the stack.
2AD2H	Add the string's length in register pair DE to the string's address in register pair HL.
2AD3H	Load register B with the last character of the string at the location of the string pointer in register pair HL.
2AD4H	Save the zero in register D at the location of the string pointer in register pair HL.
2AD5H	Exchange the string's ending address in register pair HL with the string's address on the stack.
2AD6H	Save the last character of the string in register B on the stack.
2AD7H	Load register A with the character at the location of the string pointer in register pair HL.
2AD8H - 2ADAH	Go convert the string from ASCII to binary.
2ADBH	Get the last character of the string from the stack and put it in register B.
2ADCH	Get the string's ending address from the stack and put it in register pair HL.

2ADDH	Save the character in register B at the location of the string pointer in register pair HL.
2ADEH	Return.
2ADFH - 2AE6H	**STRING ROUTINE**
2ADFH	Load register pair HL with the value of the current BASIC program pointer in register pair DE.
2AE0H - 2AE1H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a).
2AE2H	Get the return address from the stack and put it in register pair BC.
2AE3H	Get the number of bytes from the stack and put it in register pair DE.
2AE4H	Save the return address in register pair BC on the stack.
2AE5H	Load register B with the number of bytes in register E.
2AE6H	Return.
2AE7H - 2AEEH	**DISK ROUTINE**
2AE7H - 2AE8H	Check to see if the character at the location of the current BASIC program pointer in register A is equal to 7AH.
2AE9H - 2AEBH	Go to the Level II BASIC error routine and display a SN ERROR message if the character at the location of the current BASIC program pointer in register A isn't equal to 7AH.
2AECH - 2AEEH	Jump to the DOS link at 41D9H.
2AEFH - 2AF7H	**LEVEL II BASIC INP ROUTINE**
2AEFH - 2AF1H	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the port number in register A.
2AF2H - 2AF4H	Save the value of the port number in register A.
2AF5H - 2AF7H	Go get the value from the port and return.
2AF8H - 2B00H	**LEVEL II BASIC OUT ROUTINE**
2AF8H - 2AFAH	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the port number saved in memory.

2AFBH - 2AFDH	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with value to send to the port in register A.
2AFEH - 2B00H	Go send the value in register A to the port and return.
2B01H - 2B0DH	**EVALUATE EXPRESSION ROUTINE**
2B01H	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
2B02H - 2B04H	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the result in REG1.
2B05H	Save the value of the current BASIC program pointer in register pair HL on the stack.
2B06H - 2B08H	Go convert the result in REG1 to an integer.
2B09H	Load register pair DE with the integer result in register pair HL.
2B0AH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
2B0BH	Load register A with the MSB of the integer result in register D.
2B0CH	Test the value of the MSB in register A.
2B0DH	Return.
2B0EH - 2B16H	**EVALUATE EXPRESSION ROUTINE**
2B0EH - 2B10H	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the 8-bit value in register A.
2B11H - 2B13H	Save the 8-bit value in register A.
2B14H - 2B16H	Save the 8-bit value in register A.
2B17H - 2B1AH	**CHECK SYNTAX ROUTINE**
2B17H - 2B18H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a comma.
2B19H - 2B1AH	Jump.
2B1BH - 2B28H	**EVALUATE EXPRESSION ROUTINE**
2B1BH	Go bump the value of the current BASIC program

	pointer in register pair HL till it points to the next character.
2B1CH - 2B1EH	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the result in REG1.
2B1FH - 2B21H	Go convert the result in REG1 to an integer and return with the integer result in register pair DE.
2B22H - 2B24H	Go to the Level II BASIC error routine and display an FC ERROR message if the result is greater than 255.
2B25H	Decrement the value of the current BASIC program pointer in register pair HL.
2B26H	Go bump the value of the current BASIC program pointer in register pair HL until it points to the next character.
2B27H	Load register A with the 8-bit result in register E.
2B28H	Return.
2B29H - 2B2DH	**LEVEL II BASIC LLIST ROUTINE**
2B29H - 2B2AH	Load register A with the printer output device code.
2B2BH - 2B2DH	Save the value in register A as the current output device flag.
2B2EH - 2B74H	**LEVEL II BASIC LIST ROUTINE**
2B2EH	Get the return address from the stack and put it in register pair BC.
2B2FH - 2B31H	Go evaluate the range of line numbers given at the location of the current BASIC program pointer in register pair HL.
2B32H	Save the address of the first BASIC line in register pair BC on the stack.
2B33H - 2B35H	Load register pair HL with a −1.
2B36H - 2B38H	Save the value in register pair HL as the current BASIC line number.
2B39H	Get the address of the first BASIC line from the stack and put it in register pair HL.
2B3AH	Get the value of the last BASIC line number from the stack and put it in register pair DE.
2B3BH	Load register C with the LSB of the next BASIC line

	pointer at the location of the memory pointer in register pair HL.
2B3CH	Bump the value of the memory pointer in register pair HL.
2B3DH	Load register B with the MSB of the next BASIC line pointer at the location of the memory pointer in register pair HL.
2B3EH	Bump the value of the memory pointer in register pair HL.
2B3FH	Load register A with the MSB of the next BASIC line pointer in register B.
2B40H	Combine the LSB of the next BASIC line pointer in register C with the MSB of the next BASIC line pointer in register A.
2B41H - 2B43H	Jump if this is the end of the BASIC program.
2B44H - 2B46H	Go call the DOS link at 41DFH.
2B47H - 2B49H	Go scan the keyboard to see if the BREAK key or the shift @ key was pressed.
2B4AH	Save the address of the next BASIC line in register pair BC on the stack.
2B4BH	Load register C with the LSB of the BASIC line number at the location of the memory pointer in register pair HL.
2B4CH	Bump the value of the memory pointer in register pair HL.
2B4DH	Load register B with the MSB of the BASIC line number at the location of the memory pointer in register pair HL.
2B4EH	Bump the value of the memory pointer in register pair HL.
2B4FH	Save the BASIC line number in register pair BC on the stack.
2B50H	Exchange the value of the memory pointer in register pair HL with the value of the BASIC line number on the stack.
2B51H	Exchange the value of the BASIC line number in register pair HL with the value of the last BASIC line number in register pair DE.
2B52H	Compare the BASIC line number in register pair DE with the last BASIC line number in register pair HL.

2B53H	Get the value of the memory pointer from the stack and put it in register pair BC.
2B54H - 2B56H	Jump if the BASIC line number in register pair DE is greater than the last BASIC line number in register pair HL.
2B57H	Exchange the value of the last BASIC line number in register pair HL with the address of the next BASIC line on the stack.
2B58H	Save the address of the next BASIC line in register pair HL on the stack.
2B59H	Save the memory pointer in register pair BC on the stack.
2B5AH	Load register pair HL with the BASIC line number in register pair DE.
2B5BH - 2B5DH	Save the BASIC line number in register pair HL.
2B5EH - 2B60H	Go convert the BASIC line number in register pair HL to an ASCII string and send the string to the current output device.
2B61H - 2B62H	Load register A with a space.
2B63H	Get the value of the memory pointer from the stack and put it in register pair HL.
2B64H - 2B66H	Go send the space in register A to the current output device.
2B67H - 2B69H	Go move the BASIC line at the location of the memory pointer in register pair HL into the input buffer and untokenize the BASIC line.
2B6AH - 2B6CH	Load register pair HL with the starting address of the input buffer.
2B6DH - 2B6FH	Go send the BASIC line in the input buffer to the current output device.
2B70H - 2B72H	Go send a carriage return to the current output device.
2B73H - 2B74H	Loop till the listing is complete.
2B75H - 2B7DH	**DISPLAY MESSAGE ROUTINE**
2B75H	Load register A with the character at the location of the memory pointer in register pair HL.
2B76H	Check to see if the character in register A is an end of the string character.

2B77H	Return if the character in register A is an end of the string character.
2B78H - 2B7AH	Go send the character in register A to the current output device.
2B7BH	Bump the value of the memory pointer in register pair HL.
2B7CH - 2B7DH	Loop till all of the characters have been sent to the current output device.
2B7EH - 2BC5H	**UNTOKENIZE ROUTINE**
2B7EH	Save the BASIC line pointer in register pair HL on the stack.
2B7FH - 2B81H	Load register pair HL with the starting address of the input buffer.
2B82H	Load register B with the MSB of the input buffer pointer in register H.
2B83H	Load register C with the LSB of the input buffer pointer in register L.
2B84H	Get the value of the BASIC line pointer from the stack and put it in register pair HL.
2B85H - 2B86H	Load register D with the maximum length of an untokenized line.
2B87H - 2B88H	Jump.
2B89H	Bump the value of the input buffer pointer in register pair BC.
2B8AH	Decrement the character count in register D.
2B8BH	Return if 255 characters have been moved into the input buffer.
2B8CH	Load register A with the character at the location of the BASIC line pointer in register pair HL.
2B8DH	Check to see if the character in register A is an end of the BASIC line character.
2B8EH	Bump the value of the BASIC line pointer in register pair HL.
2B8FH	Save the character in register A at the location of the input buffer pointer in register pair BC.
2B90H	Return if the character in register A is an end of the BASIC line character.

2B91H - 2B93H	Jump if the character in register A isn't a BASIC token.
2B94H - 2B95H	Check to see if the character in register A is a token.
2B96H - 2B97H	Jump if the character in register A isn't a ' token.
2B98H	Decrement the value of the input buffer pointer in register pair BC.
2B99H	Decrement the value of the input buffer pointer in register pair BC.
2B9AH	Decrement the value of the input buffer pointer in register pair BC.
2B9BH	Decrement the value of the input buffer pointer in register pair BC.
2B9CH	Bump the value of the character counter in register D.
2B9DH	Bump the value of the character counter in register D.
2B9EH	Bump the value of the character counter in register D.
2B9FH	Bump the value of the character counter in register D.
2BA0H - 2BA1H	Check to see if the character in register A is an ELSE token.
2BA2H - 2BA4H	Go decrement the value of the input buffer pointer if the character in register A is an ELSE token.
2BA5H - 2BA7H	Save the value of the BASIC line pointer in register pair HL on the stack.
2BA8H	Load register E with the character in register A.
2BA9H - 2BABH	Load register pair HL with the starting address of the reserved words list.
2BACH	Load register A with the character at the location of the reserved words list pointer in register pair HL.
2BADH	Test the value of the character in register A.
2BAEH	Bump the value of the reserved words list pointer in register pair HL.
2BAFH - 2BB1H	Jump if the character at the location of the reserved words pointer in register A doesn't have bit 7 set.

2BB2H	Decrement the value of the token in register E.
2BB3H - 2BB4H	Jump if this isn't the reserved word for the token.
2BB5H - 2BB6H	Reset bit 7 of the character in register A.
2BB7H	Save the character in register A at the location of the input buffer pointer in register pair BC.
2BB8H	Bump the value of the input buffer pointer in register pair BC.
2BB9H	Decrement the value of the character counter in register D.
2BBAH - 2BBCH	Jump if the maximum number of characters have been put into the input buffer.
2BBDH	Load register A with the character at the location of the reserved words pointer in register pair HL.
2BBEH	Bump the reserved words pointer in register pair HL.
2BBFH	Test the value of the character in register A.
2BC0H - 2BC2H	Jump if bit 7 of the character in register A isn't set.
2BC3H	Get the value of the BASIC line pointer from the stack and put it in register pair HL.
2BC4H - 2BC5H	Jump.
2BC6H - 2BF4H	**LEVEL II BASIC DELETE ROUTINE**
2BC6H - 2BC8H	Go evaluate the line numbers at the location of the current BASIC program pointer in register pair HL.
2BC9H	Get the value of the last BASIC line number to be deleted from the stack and put it in register pair DE.
2BCAH	Save the address of the first BASIC line to be deleted in register pair BC on the stack.
2BCBH	Save the address of the first BASIC line to be deleted in register pair BC on the stack.
2BCCH - 2BCEH	Go get the address of the last line to be deleted.
2BCFH - 2BD0H	Jump if the last line to be deleted can't be found in the BASIC program.
2BD1H	Load register D with the MSB of the last BASIC line's address in register H.
2BD2H	Load register E with the LSB of the last BASIC line's address in register L.

2BD3H	Exchange the last BASIC line's address in register pair HL with the first BASIC line's address on the stack.
2BD4H	Save the first BASIC line's address in register pair HL on the stack.
2BD5H	Check to see if the first BASIC line's address in register pair HL is greater than or equal to the last BASIC line's address in register pair DE.
2BD6H - 2BD8H	Jump to the Level II BASIC error routine and display a FC ERROR message if the first BASIC lines address in register pair HL is greater than or equal to the last Basic line's address in register pair DE.
2BD9H - 2BDBH	Load register pair HL with the starting address of the BASIC READY message.
2BDCH - 2BDEH	Go display the BASIC READY message.
2BDFH	Get the first BASIC line's address from the stack and put it in register pair BC.
2BE0H - 2BE2H	Load register pair HL with the return address.
2BE3H	Exchange the value of the return address in register pair HL with the value of the last BASIC line's address on the stack.
2BE4H	Load register pair DE with the last BASIC line's address in register pair HL.
2BE5H - 2BE7H	Load register pair HL with the end of the BASIC program pointer.
2BE8H	Load register A with the character at the location of the memory pointer in register pair DE.
2BE9H	Save the character in register A at the location of the memory pointer in register pair BC.
2BEAH	Bump the value of the memory pointer in register pair BC.
2BEBH	Bump the value of the memory pointer in register pair DE.
2BECH	Check to see if the memory pointer in register pair DE equals the end of the BASIC program pointer in register pair HL.
2BEDH - 2BEEH	Loop till the memory pointer in register pair DE equals the end of the BASIC program pointer in register pair HL.

2BEFH	Load register H with the MSB of the memory pointer in register B.
2BF0H	Load register L with the LSB of the memory pointer in register C.
2BF1H - 2BF3H	Save the value in register pair HL as the new end of the BASIC program pointer.
2BF4H	Return.
2BF5H - 2C1EH	**LEVEL II BASIC CSAVE ROUTINE**
2BF5H - 2BF7H	Go write the leader and the sync byte on the cassette recorder.
2BF8H - 2BFAH	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the result in REG1.
2BFBH	Save the value of the current BASIC program pointer in register pair HL on the stack.
2BFCH - 2BFEH	Go get the starting address of the filename in register pair DE.
2BFFH - 2C00H	Load register A with the filename header byte.
2C01H - 2C03H	Go write the filename header byte in register A on the cassette recorder.
2C04H - 2C06H	Go write the filename header byte in register A twice more.
2C07H	Load register A with the first character of the filename at the location of the filename pointer in register pair DE.
2C08H - 2C0AH	Go write the filename in register A on the cassette recorder.
2C0BH - 2C0DH	Load register pair HL with the start of the BASIC program pointer.
2C0EH	Load register pair DE with the start of the BASIC program pointer in register pair HL.
2C0FH - 2C11H	Load register pair HL with the end of the BASIC program pointer.
2C12H	Load register A with the character at the location of the memory pointer in register pair DE.
2C13H	Bump the value of the memory pointer in register pair DE.

2C14H - 2C16H	Go write the character in register A on the cassette recorder.
2C17H	Check to see if the memory pointer in register pair DE is equal to the end of the BASIC program pointer in register pair HL.
2C18H - 2C19H	Loop till the memory pointer in register pair DE is equal to the end of the BASIC program pointer in register pair HL.
2C1AH - 2C1CH	Go turn the cassette recorder off.
2C1DH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
2C1EH	Return.
2C1FH - 2CA4H	**LEVEL II BASIC CLOAD ROUTINE**
2C1FH - 2C21H	Go turn on the cassette recorder.
2C22H	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
2C23H - 2C24H	Check to see if the character at the location of the current BASIC program pointer in register A is a ?.
2C25H - 2C26H	Jump if the character at the location of the current BASIC program pointer in register A is a ?.
2C27H	Zero register A.
2C29H	Load register A with a −1 for CLOAD?.
2C2AH	Bump the value of the current BASIC program pointer in register pair HL till it points to the next character if CLOAD?.
2C2BH	Save the CLOAD/CLOAD? flag in register A on the stack.
2C2CH	Decrement the value of the current BASIC program pointer in register pair HL.
2C2DH	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
2C2EH - 2C2FH	Zero register A.
2C30H - 2C31H	Jump if the character at the location of the current BASIC program pointer in register pair HL is an end of the BASIC statement character.

2C32H - 2C34H	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the result in REG1.
2C35H - 2C37H	Go get the filename's address in register pair DE.
2C38H	Load register A with the first character of the filename at the location of the filename pointer in register pair DE.
2C39H	Load register L with the filename in register A.
2C3AH	Get the value of the CLOAD/CLOAD? flag from the stack and put it in register A.
2C3BH	Test the value of the CLOAD/CLOAD? flag in register A.
2C3CH	Load register H with the value of the CLOAD/ CLOAD? flag in register A.
2C3DH - 2C3FH	Save the value of the CLOAD/CLOAD? flag and the filename in register pair HL in REG1.
2C40H - 2C42H	Go reinitialize the BASIC pointers if CLOAD.
2C43H - 2C45H	Load register pair HL with the CLOAD/CLOAD? flag and the filename in REG1.
2C46H	Load register pair DE with the CLOAD/CLOAD? flag and the filename in register pair HL.
2C47H - 2C48H	Load register B with the number of bytes for the filename header.
2C49H - 2C4BH	Go read a byte from the cassette recorder and return with it in register A.
2C4CH - 2C4DH	Check to see if the character in register A is a filename header byte.
2C4EH - 2C4FH	Loop if the character in register A isn't a filename header byte.
2C50H - 2C51H	Loop till three filename header bytes have been read.
2C52H - 2C54H	Go read a byte from the cassette recorder and return with it in register A.
2C55H	Bump the value of the filename in register E.
2C56H	Decrement the value of the filename in register E.
2C57H - 2C58H	Jump if no filename was specified.

2C59H	Compare the filename specified in register E with the character in register A.
2C5AH - 2C5BH	Jump if the filename specified in register E doesn't match the character in register A.
2C5CH - 2C5EH	Load register pair HL with the start of the BASIC program pointer.
2C5FH - 2C60H	Load register B with the number of zeros to look for to stop the load.
2C61H - 2C63H	Go read a byte from the cassette recorder and return with it in register A.
2C64H	Load register E with the character in register A.
2C65H	Compare the character in register A with the character at the location of the memory pointer in register pair HL.
2C66H	Combine the result in register A with the value of the CLOAD/CLOAD? flag in register D.
2C67H - 2C68H	Jump if CLOAD? and the bytes don't match.
2C69H	Save the character in register E at the location of the memory pointer in register pair HL.
2C6AH - 2C6CH	Go check for an OM ERROR.
2C6DH	Load register A with the character at the location of the memory pointer in register pair HL.
2C6EH	Check to see if the byte just read in register A is equal to zero.
2C6FH	Bump the value of the memory pointer in register pair HL.
2C70H - 2C71H	Loop if the byte in register A isn't equal to zero.
2C72H - 2C74H	Go blink the asterisk on the video display.
2C75H - 2C76H	Loop till three zeros in a row have been read from the cassette recorder.
2C77H - 2C79H	Save the value of the memory pointer in register pair HL as the new end of the BASIC program pointer.
2C7AH - 2C7CH	Load register pair HL with the starting address of the BASIC READY message.
2C7DH - 2C7FH	Go display the BASIC READY message.
2C80H - 2C82H	Go turn off the cassette recorder.

2C83H - 2C85H	Load register pair HL with the start of the BASIC program pointer.
2C86H	Save the start of the BASIC program pointer in register pair HL on the stack.
2C87H - 2C89H	Jump.
2C8AH - 2C8CH	Load register pair HL with the starting address of the BAD message.
2C8DH - 2C8FH	Go display the BAD message.
2C90H - 2C92H	Jump.
2C93H - 2C95H	Go display the filename on the video display.
2C96H - 2C97H	Load register B with the number of zeros to be found to stop the search.
2C98H - 2C9AH	Go read a byte from the cassette recorder and return with it in register A.
2C9BH	Check to see if the character in register A is equal to zero.
2C9CH - 2C9DH	Loop if the character in register A isn't equal to zero.
2C9EH - 2C9FH	Loop till three zeros in a row have been read from the cassette recorder.
2CA0H - 2CA2H	Go read the leader and the sync byte on the cassette recorder.
2CA3H - 2CA4H	Jump.
2CA5H - 2CA9H	**MESSAGE STORAGE LOCATION**
2CA5H - 2CA9H	The BAD message is stored here.
2CAAH - 2CB0H	**LEVEL II BASIC PEEK ROUTINE**
2CAAH - 2CACH	Go convert the current result in REG1 to an integer and return with it in register pair HL.
2CADH	Load register A with the value at the location of the memory pointer in register pair HL.
2CAEH - 2CB0H	Go save the 8-bit value in register A as the current result in REG1.
2CB1H - 2CBCH	**LEVEL II BASIC POKE ROUTINE**
2CB1H - 2CB3H	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the integer result in register pair DE.

2CB4H	Save the value of the memory pointer in register pair DE on the stack.
2CB5H - 2CB6H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a comma.
2CB7H - 2CB9H	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the 8-bit value in register A.
2CBAH	Get the value of the memory pointer from the stack and put it in register pair DE.
2CBBH	Save the value in register A at the location of the memory pointer in register pair DE.
2CBCH	Return.
2CBDH - 2E52H	**LEVEL II BASIC USING ROUTINE**
2CBDH - 2CBFH	Go evaluate the string expression at the location of the current BASIC program pointer in register pair HL.
2CC0H - 2CC2H	Go make sure the expression that was just evaluated was a string.
2CC3H - 2CC4H	Go check the syntax. The character at the location of the current BASIC program pointer in register pair HL must be a semicolon.
2CC5H	Load register pair DE with the value of the current BASIC program pointer in register pair HL.
2CC6H - 2CC8H	Load register pair HL with the USING string's VARPTR.
2CC9H - 2CCAH	Jump.
2CCBH - 2CCDH	Load register A with the value of the READ/INPUT flag.
2CCEH	Check to see if the READ/INPUT flag in register A indicates INPUT.
2CCFH - 2CD0H	Jump if the READ/INPUT flag in register A indicates INPUT.
2CD1H	Get the value of the current BASIC program pointer from the stack and put it in register pair DE.
2CD2H	Load register pair HL with the value of the current BASIC program pointer in register pair DE.
2CD3H	Save the USING string's VARPTR in register pair HL on the stack.

2CD4H	Zero register A.
2CD5H - 2CD7H	Save the value in register A as the READ/INPUT flag.
2CD8H	Check to see if the value in register pair DE is equal to zero.
2CD9H	Save the value in register pair AF on the stack.
2CDAH	Save the value in register pair DE on the stack.
2CDBH	Load register B with the USING string's length at the location of the USING string's VARPTR in register pair HL.
2CDCH	Check to see if the USING string's length in register B is equal to zero.
2CDDH - 2CDFH	Go to the Level II BASIC error routine and display a FC ERROR message if the USING string's length in register B is equal to zero.
2CE0H	Bump the value of the USING string's VARPTR in register pair HL.
2CE1H	Load register C with the LSB of the USING string's address at the location of the USING string's VARPTR in register pair HL.
2CE2H	Bump the value of the USING string's VARPTR in register pair HL.
2CE3H	Load register H with the MSB of the USING string's address at the location of the USING string's VARPTR in register pair HL.
2CE4H	Load register L with the LSB of the USING string's address in register C.
2CE5H - 2CE6H	Jump.
2CE7H	Load register E with the USING string's length in register B.
2CE8H	Save the value of the current USING string pointer in register pair HL.
2CE9H - 2CEAH	Load register C with the number of %'s substring length.
2CEBH	Load register A with the character at the location of the USING string pointer in register pair HL.
2CECH	Bump the value of the USING string pointer in register pair HL.

2CEDH - 2CEEH		Check to see if the character in register A is a %.
2CEFH - 2CF1H		Jump if the character in register A is a %.
2CF2H - 2CF3H		Check to see if the character in register A is a space.
2CF4H - 2CF5H		Jump if the character in register A isn't a space.
2CF6H		Bump the % substring length in register C.
2CF7H - 2CF8H		Decrement the USING string's length in register B.
2CF9H		Get the value of the USING string pointer from the stack and put it in register pair HL.
2CFAH		Load register B with the USING string's length.
2CFBH - 2CFCH		Load register A with a %.
2CFDH - 2CFFH		Go print a + if necessary.
2D00H - 2D02H		Go print the character in register A.
2D03H		Zero register A.
2D04H		Load register E with the value in register A.
2D05H		Load register D with the value in register A.
2D06H - 2D08H		Go print a + if necessary.
2D09H		Load register D with the value in register A.
2D0AH		Load register A with the character at the location of the USING string pointer in register pair HL.
2D0BH		Bump the value of the USING string pointer in register pair HL.
2D0CH - 2D0DH		Check to see if the character in register A is a !.
2D0EH - 2D10H		Jump if the character in register A is a !.
2D11H - 2D12H		Check to see if the character in register A is a #.
2D13H - 2D14H		Jump if the character in register A is a #.
2D15H		Decrement the value of the string's length in register B.
2D16H - 2D18H		Jump if this is the end of the USING string.
2D19H - 2D1AH		Check to see if the character in register A is a +.
2D1BH - 2D1CH		Set the leading + flag in register A.
2D1DH - 2D1EH		Jump if the character in register A was a +.

2D1FH	Decrement the value of the USING string pointer in register pair HL.
2D20H	Load register A with the character at the location of the USING string pointer in register pair HL.
2D21H	Bump the value of the USING string pointer in register pair HL.
2D22H - 2D23H	Check to see if the character in register A is a decimal point.
2D24H - 2D25H	Jump if the character in register A is a decimal point.
2D26H - 2D27H	Check to see if the character in register A is a %.
2D28H - 2D29H	Jump if the character in register A is a %.
2D2AH	Compare the character in register A with the character at the location of the USING string pointer in register pair HL.
2D2BH - 2D2CH	Jump if the character in register A isn't equal to the character at the location of the USING string pointer in register pair HL.
2D2DH - 2D2EH	Check to see if the character in register A is a $.
2D2FH - 2D30H	Jump if the character in register A is a $.
2D31H - 2D32H	Check to see if the character in register A is a *.
2D33H - 2D34H	Jump if the character in register A isn't a *.
2D35H	Load register A with the USING string's length in register B.
2D36H - 2D37H	Check to see if the USING string's length in register A is at least two.
2D38H	Bump the value of the USING string pointer in register pair HL.
2D39H - 2D3AH	Jump if the USING string's length in register A isn't at least two.
2D3BH	Load register A with the character at the location of the USING string pointer in register pair HL.
2D3CH - 2D3DH	Check to see if the character in register A is a $.
2D3EH - 2D3FH	Set the ** edit flag in register A.
2D40H - 2D41H	Jump if the character in register A isn't a $.
2D42H	Decrement the value of the USING string's length in register B.

2D43H — Bump the number of characters to the left of the decimal point in register E.

2D46H - 2D47H — Mask the value of the edit flag in register A for leading $'s.

2D48H — Bump the value of the USING string pointer in register pair HL.

2D49H — Bump the number of characters to the left of the decimal point in register E.

2D4AH — Combine the value of the edit flag in register D with the value of the edit flag in register A.

2D4BH — Load register D with the value of the edit flag in register A.

2D4CH — Bump the number of characters to the left of the decimal point in register E.

2D4DH - 2D4EH — Zero the number of characters to the right of the decimal point in register C.

2D4FH — Decrement the value of the string's length in register B.

2D50H - 2D51H — Jump if this is the end of the string.

2D52H — Load register A with the character at the location of the USING string pointer in register pair HL.

2D53H — Bump the value of the USING string pointer in register pair HL.

2D54H - 2D55H — Check to see if the character in register A is a decimal point.

2D56H - 2D57H — Jump if the character in register A is a decimal point.

2D58H - 2D59H — Check to see if the character in register A is a #.

2D5AH - 2D5BH — Jump if the character in register A is a #.

2D5CH - 2D5DH — Check to see if the character in register A is a comma.

2D5EH - 2D5FH — Jump if the character in register A isn't a comma.

2D60H — Load register A with the value of the edit flag in register D.

2D61H - 2D62H — Mask the edit flag in register A for commas.

2D63H — Load register D with the value of the edit flag in register A.

2D64H - 2D65H	Jump.
2D66H	Load register A with the character at the location of the USING string pointer in register pair HL.
2D67H - 2D68H	Check to see if the character in register A is a #.
2D69H - 2D6AH	Load register A with a decimal point.
2D6BH - 2D6CH	Jump if the character in register A wasn't a #.
2D6DH - 2D6EH	Load register C with the number of characters to the right of the decimal point.
2D6FH	Bump the value of the USING string pointer in register pair HL.
2D70H	Bump the number of characters to the right of the decimal point in register C.
2D71H	Decrement the value of the string's length in register B.
2D72H - 2D73H	Jump if this is the end of the string.
2D74H	Load register A with the character at the location of the USING string pointer in register pair HL.
2D75H	Bump the value of the USING string pointer in register pair HL.
2D76H - 2D77H	Check to see if the character in register A is a #.
2D78H - 2D79H	Jump if the character in register A is a #.
2D7AH	Save the value of the edit flag and the count of the characters to the left of the decimal point in register pair DE on the stack.
2D7BH - 2D7DH	Load register pair DE with the return address.
2D7EH	Save the value of the return address in register pair DE on the stack.
2D7FH	Load register D with the MSB of the USING string pointer in register H.
2D80H	Load register E with the LSB of the USING string pointer in register L.
2D81H - 2D82H	Check to see if the character in register A is an up arrow.
2D83H	Return if the character in register A isn't an up arrow.
2D84H	Check to see if the character at the location of the

	USING string pointer in register pair HL is an up arrow.
2D85H	Return if the character at the location of the USING string pointer in register pair HL isn't an up arrow.
2D86H	Bump the value of the USING string pointer in register pair HL.
2D87H	Check to see if there is a third up arrow at the location of the USING string pointer in register pair HL.
2D88H	Return if the character at the location of the USING string pointer in register pair HL isn't an up arrow.
2D89H	Bump the value of the USING string pointer in register pair HL.
2D8AH	Check to see if the character at the location of the USING string pointer in register pair HL is a fourth up arrow.
2D8BH	Return if the character at the location of the USING string pointer in register pair HL isn't an up arrow.
2D8CH	Bump the value of the USING string pointer in register pair HL.
2D8DH	Load register A with the value of the USING string's length in register B.
2D8EH - 2D8FH	Check to see if this is the end of the USING string.
2D90H	Return if there isn't at least four characters left in the USING string.
2D91H	Clean up the stack.
2D92H	Get the edit flag and the count of the characters to the left of the decimal point from the stack and put it in register pair DE.
2D93H	Load register B with the USING string's length in register A.
2D94H	Set the exponential notation flag in register D.
2D95H	Bump the value of the USING string pointer in register pair HL.
2D97H	Load register pair HL with the value of the USING string pointer in register pair DE.
2D98H	Get the edit flag and the count of the characters to the left of the decimal point from the stack and put it in register pair DE.

2D99H	Load register A with the value of the edit flag in register D.
2D9AH	Decrement the value of the USING string pointer in register pair HL.
2D9BH	Bump the number of characters to the left of the decimal point in register E.
2D9CH - 2D9DH	Check to see if the sign flag is set in register A.
2D9EH - 2D9FH	Jump if the sign flag has already been processed.
2DA0H	Decrement the number of characters to the left of the decimal point in register E.
2DA1H	Load register A with the USING string's length in register B.
2DA2H	Check to see if this is the end of the string.
2DA3H - 2DA4H	Jump if this is the end of the string.
2DA5H	Load register A with the character at the location of the USING string pointer in register pair HL.
2DA6H - 2DA7H	Check to see if the character in register A is a –.
2DA8H - 2DA9H	Jump if the character in register A is a –.
2DAAH - 2DABH	Check to see if the character in register A is a +.
2DACH - 2DADH	Jump if the character in register A isn't a +.
2DAEH - 2DAFH	Set the edit flag for a + character.
2DB0H - 2DB1H	Set the edit flag for a – character.
2DB2H	Combine the value of the edit flag in register D with the value of the edit flag in register A.
2DB3H	Load register D with the value of the edit flag in register A.
2DB4H	Decrement the value of the USING string's length in register B.
2DB5H	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
2DB6H	Load register A with the character at the location of the current BASIC program pointer in register pair HL.
2DB7H - 2DB8H	Jump if this is the end of the BASIC statement.
2DB9H	Save the count of the characters to the right of the

	decimal point and the USING string's length in register pair BC on the stack.
2DBAH	Save the edit flag and the number of characters to the left of the decimal point in register pair DE on the stack.
2DBBH - 2DBDH	Go evaluate the expression at the location of the current BASIC program pointer and return with the result in REG1.
2DBEH	Get the edit flag and the number of characters to the left of the decimal point from the stack and put it in register pair DE.
2DBFH	Get the number of characters to the right of the decimal point and the USING string's length from the stack and put it in register pair BC.
2DC0H	Save the number of characters to the right of the decimal point and the USING string's length in register pair BC on the stack.
2DC1H	Save the value of the current BASIC program pointer in register pair HL on the stack.
2DC2H	Load register B with the number of characters to the left of the decimal point in register E.
2DC3H	Load register A with the number of characters to the left of the decimal point in register B.
2DC4H	Add the number of characters to the right of the decimal point in register C to the number of characters to the left of the decimal point in register A.
2DC5H - 2DC6H	Check to see if the total number of characters in register A is greater than 24.
2DC7H - 2DC9H	Go to the Level II BASIC error routine and display a FC ERROR message if the total number of characters in register A is greater than 24.
2DCAH	Load register A with the value of the edit flag in register D.
2DCBH - 2DCCH	Set the edit flag in register A.
2DCDH - 2DCFH	Go convert the result in REG1 to an ASCII string.
2DD0H - 2DD2H	Go display the ASCII string.
2DD3H	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
2DD4H	Decrement the value of the current BASIC program pointer in register pair HL.

2DD5H	Go bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
2DD6H	Set the Carry flag.
2DD7H - 2DD8H	Jump if the character at the location of the current BASIC program pointer in register A is an end of the BASIC statement character.
2DD9H - 2DDBH	Save the character at the location of the current BASIC program pointer in register A.
2DDCH - 2DDDH	Check to see if the character at the location of the current BASIC program pointer in register A is a semicolon.
2DDEH - 2DDFH	Jump if the character at the location of the current BASIC program pointer in register A is a semicolon.
2DE0H - 2DE1H	Check to see if the character at the location of the current BASIC program pointer in register A is a comma.
2DE2H - 2DE4H	Go to the Level II BASIC error routine and display an SN ERROR message if the character at the location of the current BASIC program pointer in register A isn't a comma.
2DE5H	Bump the value of the current BASIC program pointer in register pair HL till it points to the next character.
2DE6H	Get the USING string's length from the stack and put it in register B.
2DE7H	Load register pair DE with the value of the current BASIC program pointer in register pair HL.
2DE8H	Get the USING string's VARPTR from the stack and put it in register pair HL.
2DE9H	Save the USING string's VARPTR in register pair HL on the stack.
2DEAH	Save the character at the location of the current BASIC program pointer in register A on the stack.
2DEBH	Save the value of the current BASIC program pointer in register pair DE on the stack.
2DECH	Load register A with the USING string's length at the location of the USING string's VARPTR in register pair HL.
2DEDH	Compare the USING string's length in register B with the USING string's length in register A.

2DEEH	Bump the value of the USING string's VARPTR in register pair HL.
2DEFH	Load register C with the LSB of the USING string's address at the location of the USING string's VARPTR in register pair HL.
2DF0H	Bump the value of the USING string's VARPTR in register pair HL.
2DF1H	Load register H with the MSB of the USING string's address at the location of the USING string's VARPTR in register pair HL.
2DF2H	Load register L with the LSB of the USING string's address in register C.
2DF3H - 2DF4H	Zero register D.
2DF5H	Load register E with the USING string's offset in register A.
2DF6H	Add the USING string's offset in register pair DE to the USING string's address in register pair HL.
2DF7H	Load register A with the USING string's length in register B.
2DF8H	Check to see if this is the end of the USING string.
2DF9H - 2DFBH	Jump if this isn't the end of the USING string.
2DFCH - 2DFDH	Jump if this is the end of the USING string.
2DFEH - 2E00H	Go print a + if necessary.
2E01H - 2E03H	Go send the character in register A to the current output device.
2E04H	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
2E05H	Get the character at the location of the current BASIC program pointer from the stack and put it in register A.
2E06H - 2E08H	Jump if this isn't the end of the BASIC statement.
2E09H - 2E0BH	Go send a carriage return to the current output device if necessary.
2E0CH	Exchange the value of the current BASIC program pointer in register pair HL with the value of the USING string's VARPTR on the stack.
2E0DH - 2E0FH	Go get the USING string's address in register pair DE.

2E10H	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
2E11H - 2E13H	Return.
2E14H - 2E15H	Load register C with the number of characters to be printed.
2E17H	Clean up the stack.
2E18H	Decrement the value of the string's length in register B.
2E19H - 2E1BH	Go print a + if necessary.
2E1CH	Get the value of the current BASIC program pointer from the stack and put it in register pair HL.
2E1DH	Get the character at the location of the current BASIC program pointer from the stack and put it in register A.
2E1EH - 2E1FH	Jump if the character at the location of the current BASIC program pointer in register A is an end of the BASIC statement character.
2E20H	Save the USING string's length and the number of characters to be printed in register pair BC on the stack.
2E21H - 2E23H	Go evaluate the expression at the location of the current BASIC program pointer in register pair HL and return with the result in REG1.
2E24H - 2E26H	Go make sure the current result in REG1 is a string.
2E27H	Get the USING string's length and the number of characters to be printed from the stack and put it in register pair BC.
2E28H	Save the USING string's length and the number of characters to be printed in register pair BC on the stack.
2E29H	Save the value of the current BASIC program pointer in register pair HL on the stack.
2E2AH - 2E2CH	Load register pair HL with the string's VARPTR in REG1.
2E2DH	Load register B with the number of characters to be printed in register C.
2E2EH - 2E2FH	Zero register C.
2E30H	Save the length of the string to be printed in register B on the stack.

2E31H - 2E33H	Go evaluate the string to be printed.
2E34H - 2E36H	Go send the string to the current output device.
2E37H - 2E39H	Load register pair HL with the string's VARPTR in REG1.
2E3AH	Get the length of the string to be printed from the stack and put it in register A.
2E3BH	Subtract the string's length at the location of the string's VARPTR in register pair HL from the length of the string to be printed in register A.
2E3CH	Load register B with the number of spaces to be printed in register A.
2E3DH - 2E3EH	Load register A with a space.
2E3FH	Bump the number of spaces in register B.
2E40H	Decrement the number of spaces in register B.
2E41H - 2E43H	Jump if all of the spaces have been printed.
2E44H - 2E46H	Go send a space to the current output device.
2E47H - 2E48H	Jump.
2E49H	Save the value in register A on the stack.
2E4AH	Load register A with the value in register D.
2E4BH	Check to see if register A is equal to zero.
2E4CH - 2E4DH	Load register A with a +.
2E4EH - 2E50H	Go send a + to the current output device if register D is equal to zero.
2E51H	Get the value from the stack and put it in register A.
2E52H	Return.
2E53H - 2FFAH	**LEVEL II BASIC EDIT ROUTINE**
2E53H - 2E55H	Reset the error flag.
2E56H - 2E58H	Load register pair HL with the error line number.
2E59H	Load register A with the MSB of the error line number in register H.
2E5AH	Combine the LSB of the error line number in register L with the MSB of the error line number in register A.

2E5BH	Bump the combined value of the error line number in register A.
2E5CH	Load register pair DE with the error line number in register pair HL.
2E5DH	Return if Level II BASIC is in the command mode.
2E5EH - 2E5FH	Jump.
2E60H - 2E62H	Go evaluate the line number at the location of the current BASIC program pointer in register pair HL and return with it in register pair DE.
2E63H	Return if there wasn't a line number.
2E64H	Clean up the stack.
2E65H	Load register pair HL with the line number to be edited in register pair DE.
2E66H - 2E68H	Save the value of the line number to be edited in register pair HL.
2E69H	Load register pair DE with the line number to be edited in register pair HL.
2E6AH - 2E6CH	Go find the address of the BASIC line number in register pair DE and return with it in register pair BC.
2E6DH - 2E6FH	Go to the Level II BASIC error routine and display an UL ERROR message if the BASIC line number doesn't exist.
2E70H	Load register H with the MSB of the memory pointer in register B.
2E71H	Load register L with the LSB of the memory pointer in register C.
2E72H	Bump the value of the memory pointer in register pair HL.
2E73H	Bump the value of the memory pointer in register pair HL.
2E74H	Load register C with the LSB of the BASIC line number at the location of the memory pointer in register pair HL.
2E75H	Bump the value of the memory pointer in register pair HL.
2E76H	Load register B with the MSB of the BASIC line number at the location of the memory pointer in register pair HL.

2E77H	Bump the value of the memory pointer in register pair HL.
2E78H	Save the value of the BASIC line number in register pair BC on the stack.
2E79H - 2E7BH	Go move the BASIC line into the input buffer and expand it.
2E7CH	Get the value of the BASIC line number from the stack and put it in register pair HL.
2E7DH	Save the value of the BASIC line number in register pair HL on the stack.
2E7EH - 2E80H	Go convert the BASIC line number in register pair HL to an ASCII string and display it on the video display.
2E81H - 2E82H	Load register A with a space.
2E83H - 2E85H	Go display the space in register A.
2E86H - 2E88H	Load register pair HL with the starting address of the input buffer.
2E89H - 2E8AH	Load register A with the turn on the cursor character.
2E8BH - 2E8DH	Go turn on the cursor.
2E8EH	Save the value of the input buffer pointer in register pair HL on the stack.
2E8FH - 2E90H	Load register C with the number of characters examined so far.
2E91H	Bump the number of characters examined so far in register C.
2E92H	Load register A with the character at the location of the input buffer pointer in register pair HL.
2E93H	Check to see if the character in register A is an end of the BASIC line character.
2E94H	Bump the value of the input buffer pointer in register pair HL.
2E95H - 2E96H	Loop till the end of the BASIC line has been found.
2E97H	Get the value of the input buffer pointer from the stack and put it in register pair HL.
2E98H	Zero register B.
2E99H - 2E9AH	Zero register D.

2E9BH - 2E9DH	Go scan the keyboard.
2D9EH - 2E9FH	Adjust the value in register A for a numeric character.
2EA0H - 2EA1H	Jump if the character in register A is alphabetic.
2EA2H - 2EA3H	Check to see if the character is register A is numeric.
2EA4H - 2EA5H	Jump if the character in register A isn't numeric.
2EA6H	Load register E with the binary value of the character in register A.
2EA7H	Load register A with the value in register D.
2EA8H	Multiply the value in register A by two.
2EA9H	Multiply the value in register A by two.
2EAAH	Add the value in register D to the value in register A.
2EABH	Multiply the value in register A by two.
2EACH	Add the value in register E to the value in register A.
2EADH	Load register D with the value in register A.
2EAEH - 2EAFH	Loop till a nonnumeric character is pressed.
2EB0H	Save the value of the input buffer pointer in register pair HL on the stack.
2EB1H - 2EB3H	Load register pair HL with the return address.
2EB4H	Exchange the value of the return address in register pair HL with the value of the input buffer pointer on the stack.
2EB5H	Decrement the numeric value in register D.
2EB6H	Bump the numeric value in register D.
2EB7H - 2EB9H	Jump if the numeric value in register D isn't equal to zero.
2EBAH	Load register D with a one.
2EBBH - 2EBCH	Check to see if the character in register A is a backspace character.
2EBDH - 2EBFH	Jump if the character in register A is a backspace character.
2EC0H - 2EC1H	Check to see if the character in register A is a carriage return.

2EC2H	- 2EC4H	Jump if the character in register A is a carriage return.
2EC5H	- 2EC6H	Check to see if the character in register A is a space.
2EC7H	- 2EC8H	Jump if the character in register A is a space.
2EC9H	- 2ECAH	Check to see if the character in register A is lower-case.
2ECBH	- 2ECCH	Jump if the character in register A isn't lowercase.
2ECDH	- 2ECEH	Convert the lowercase character in register A to uppercase.
2ECFH	- 2ED0H	Check to see if the character in register A is a Q.
2ED1H	- 2ED3H	Jump if the character in register A is a Q.
2ED4H	- 2ED5H	Check to see if the character in register A is an L.
2ED6H	- 2ED7H	Jump if the character in register A is an L.
2ED9H	- 2EDAH	Check to see if the character in register A is an S.
2EDBH	- 2EDCH	Jump if the character in register A is an S.
2EDDH	- 2EDEH	Check to see if the character in register A is an I.
2EDFH	- 2EE1H	Jump if the character in register A is an I.
2EE2H	- 2EE3H	Check to see if the character in register A is a D.
2EE4H	- 2EE6H	Jump if the character in register A is a D.
2EE7H	- 2EE8H	Check to see if the character in register A is a C.
2EE9H	- 2EEBH	Jump if the character in register A is a C.
2EECH	- 2EEDH	Check to see if the character in register A is an E.
2EEEH	- 2EF0H	Jump if the character in register A is an E.
2EF1H	- 2EF2H	Check to see if the character in register A is an X.
2EF3H	- 2EF5H	Jump if the character in register A is an X.
2EF6H	- 2EF7H	Check to see if the character in register A is a K.
2EF8H	- 2EF9H	Jump if the character in register A is a K.
2EFAH	- 2EFBH	Check to see if the character in register A is an H.
2EFCH	- 2EFEH	Jump if the character in register A is an H.
2EFFH	- 2F00H	Check to see if the character in register A is an A.

2F01H	Return if the character in register A isn't an A.
2F02H	Clean up the stack.
2F03H	Get the BASIC line number from the stack and put it in register pair DE.
2F04H - 2F06H	Go print a carriage return on the video display if necessary.
2F07H - 2F09H	Jump.
2F0AH	Load register A with the character at the location of the input buffer pointer in register pair HL.
2F0BH	Check to see if the character in register A is an end of the BASIC line character.
2F0CH	Return if the character in register A is an end of the BASIC line character.
2F0DH	Bump the character position in register B.
2F0EH - 2F10H	Go display the character in register A.
2F11H	Bump the value of the input buffer pointer in register pair HL.
2F12H	Decrement the number of times to perform the operation in register D.
2F13H - 2F14H	Loop until done.
2F15H	Return.
2F16H	Save the value of the input buffer pointer in register pair HL on the stack.
2F17H - 2F19H	Load register pair HL with the return address.
2F1AH	Exchange the value of the return address in register pair HL with the value of the input buffer pointer on the stack.
2F1BH	Set the KILL/SEARCH flag for KILL.
2F1CH	Save the KILL/SEARCH flag on the stack.
2F1DH - 2F1FH	Go scan the keyboard for the character to locate.
2F20H	Load register E with the character to locate in register A.
2F21H	Get the KILL/SEARCH flag from the stack.
2F22H	Save the KILL/SEARCH flag on the stack.

2F23H - 2F25H If KILL then go print a !.

2F26H Load register A with the character at the location of the input buffer pointer in register pair HL.

2F27H Check to see if the character in register A is an end of the BASIC line character.

2F28H - 2F2AH Jump if the character in register A is an end of the BASIC line character.

2F2BH - 2F2DH Go display the character in register A.

2F2EH Get the KILL/SEARCH flag from the stack.

2F2FH Save the KILL/SEARCH flag on the stack.

2F30H - 2F32H Go delete the character from the input buffer if KILL.

2F33H - 2F34H Jump if KILL.

2F35H Bump the value of the current input buffer pointer in register pair HL.

2F36H Bump the value of the character position counter in register B.

2F37H Load register A with the character at the location of the current input buffer pointer in register pair HL.

2F38H Check to see if the character in register A is the same as the character to be located in register E.

2F39H - 2F3AH Loop till the character to be located is found.

2F3BH Decrement the number of times to perform the operation in register D.

2F3CH - 2F3DH Loop until done.

2F3EH Get the KILL/SEARCH flag from the stack.

2F3FH Return.

2F40H - 2F42H Go display the line being edited.

2F43H - 2F45H Go display a carriage return if necessary.

2F46H Get the BASIC line number from the stack and put it in register pair BC.

2F47H - 2F49H Jump.

2F4AH Load register A with the character at the location of the input buffer pointer in register pair HL.

2F4BH	Check to see if the character in register A is an end of the BASIC line character.
2F5CH	Return if the character in register A is an end of the BASIC line character.
2F4DH - 2F4EH	Load register A with an !.
2F4FH - 2F51H	Go display the ! in register A.
2F52H	Load register A with the character at the location of the input buffer pointer in register pair HL.
2F53H	Check to see if the character in register A is an end of the BASIC line character.
2F54H - 2F55H	Jump if the character in register A is an end of the BASIC line character.
2F56H - 2F58H	Go display the character in register A.
2F59H - 2F5BH	Go delete the character from the input buffer.
2F5CH	Decrement the number of times to perform the operation in register D.
2F5DH - 2F5EH	Loop until done.
2F5FH - 2F60H	Load register A with an !.
2F61H - 2F63H	Go display the ! in register A.
2F64H	Return.
2F65H	Load register A with the character at the location of the input buffer pointer in register pair HL.
2F66H	Check to see if the character in register A is an end of the BASIC line character.
2F67H	Return if the character in register A is an end of the BASIC line character.
2F68H - 2F6AH	Go get the character to put in the input buffer from the keyboard.
2F6BH	Save the character in register A at the location of the input buffer pointer in register pair HL.
2F6CH - 2F6EH	Go display the character in register A.
2F6FH	Bump the value of the input buffer pointer in register pair HL.
2F70H	Bump the character position in register B.

2F71H	Decrement the number of times to perform the operation in register D.
2F72H - 2F73H	Loop until done.
2F74H	Return.
2F75H - 2F76H	Zero the location of the input buffer pointer in register pair HL.
2F77H	Load register C with the character position in register B.
2F78H - 2F79H	Load register D with the number of times to perform the operation.
2F7AH - 2F7CH	Go display the register BASIC line if necessary.
2F7DH - 2F7FH	Go get the character to be inserted from the keyboard.
2F80H	Check to see if a key was pressed.
2F81H - 2F83H	Loop until a key is pressed.
2F84H - 2F85H	Check to see if the character in register A is a backspace character.
2F86H - 2F87H	Jump if the character in register A is a backspace character.
2F88H - 2F89H	Check to see if the character in register A is a carriage return.
2F8AH - 2F8CH	Jump if the character in register A is a carriage return.
2F8DH - 2F8EH	Check to see if the character in register A is a shift up arrow.
2F8FH	Return if the character in register A is shift up arrow.
2F90H - 2F91H	Jump.
2F92H - 2F93H	Load register A with a backspace the cursor character.
2F94H	Decrement the character position in register B.
2F95H	Bump the character position in register B.
2F96H - 2F97H	Jump if this is the first character of the BASIC line.
2F98H - 2F9AH	Go backspace the cursor on the video display.

2F9BH	Descrement the value of the input buffer pointer in register pair HL.
2F9CH	Decrement the character position in register B.
2F9DH - 2F9FH	Load register pair DE with the return address.
2FA0H	Save the value of the return address in register pair DE on the stack.
2FA1H	Save the value of the input buffer pointer in register pair HL on the stack.
2FA2H	Decrement the character position in register C.
2FA3H	Load register A with the character at the location of the input buffer pointer in register pair HL.
2FA4H	Check to see if the character in register A is an end of the BASIC line character.
2FA5H	Set the Carry flag.
2FA6H - 2FA8H	Jump if the character in register A is an end of the BASIC line character.
2FA9H	Bump the value of the input buffer pointer in register pair HL.
2FAAH	Load register A with the character at the location of the input buffer pointer in register pair HL.
2FABH	Decrement the value of the input buffer pointer in register pair HL.
2FACH	Save the character in register A at the location of the current input buffer pointer in register pair HL.
2FADH	Bump the value of the input buffer pointer in register pair HL.
2FAEH - 2FAFH	Loop until the line has been moved down.
2FB0H	Save the character to be inserted in register A on the stack.
2FB1H	Load register A with the number of characters in the input buffer in register C.
2FB2H - 2FB3H	Check for the maximum BASIC line length.
2FB4H - 2FB5H	Jump if the maximum BASIC line length hasn't been reached.

2FB6H	Get the character to be inserted from the stack and put it in register A.
2FB7H - 2FB8H	Jump.
2FB9H	Subtract the character position in register B from the number of characters in the input buffer in register A.
2FBAH	Bump the number of characters in the input buffer in register C.
2FBBH	Bump the character position in register B.
2FBCH	Save the character position and the number of characters in the input buffer in register pair BC on the stack.
2FBDH	Load register pair DE with the input buffer pointer in register pair HL.
2FBEH	Load register L with the character count in register A.
2FBFH - 2FC0H	Zero register H.
2FC1H	Add the value of the input buffer pointer in register pair DE to the character count in register pair HL.
2FC2H	Load register B with the MSB of the end of the BASIC line pointer in register H.
2FC3H	Load register C with the LSB of the end of the BASIC line pointer in register L.
2FC4H	Bump the value of the end of the BASIC line pointer in register pair HL.
2FC5H - 2FC7H	Go move the BASIC line up one character.
2FC8H	Get the character position and the number of characters in the input buffer from the stack and put it in register pair BC.
2FC9H	Get the character to be inserted from the stack and put it in register A.
2FCAH	Save the character in register A at the location of the current input buffer pointer in register pair HL.
2FCBH - 2FCDH	Go display the character in register A.
2FCEH	Bump the value of the input buffer pointer in register pair HL.
2FCFH - 2FD1H	Jump.

2FD2H	Load register A with the number of times to backspace in register B.
2FD3H	Check to see if this is the start of the BASIC line.
2FD4H	Return if this is the start of the BASIC line.
2FD5H	Decrement the character position in register B.
2FD6H	Decrement the value of the input buffer pointer in register pair HL.
2FD7H - 2FD8H	Load register A with a backspace the cursor character.
2FD9H - 2FDBH	Go backspace the cursor on the video display.
2FDCH	Decrement the number of times to perform the operation in register D.
2FDDH - 2FDEH	Loop until done.
2FDFH	Return.
2FE0H - 2FE2H	Go display the rest of the BASIC line.
2FE3H - 2FE5H	Go display a carriage return if necessary.
2FE6H	Clean up the stack.
2FE7H	Get the BASIC line number from the stack and put it in register pair DE.
2FE8H	Load register A with the MSB of the BASIC line number in register D.
2FE9H	Combine the LSB of the BASIC line number in register E with the MSB of the BASIC line number in register A.
2FEAH	Bump the combined BASIC line number in register A.
2FEBH - 2FEDH	Load register pair HL with the starting address of the input buffer.
2FEEH	Decrement the value of the input buffer pointer in register pair HL.
2FEFH	Return if this is the Level II BASIC command mode.
2FF0H	Set the Carry flag.
2FF1H	Bump the value of the input buffer pointer in register pair HL.

2FF2H	Save the command mode flag in register pair AF on the stack.
2FF3H - 2FF5H	Jump to the Level II BASIC READY routine.
2FF6H	Clean up the stack.
2FF7H	Get the BASIC line number from the stack and put it in register pair DE.
2FF8H - 2FFAH	Jump to the Level II BASIC READY routine.
2FFBH - 2FFFH	**THE END OF THE LEVEL II BASIC ROMS**
2FFBH - 2FFFH	Nothing here.
3000H - 37DDH	**THIS SPACE IS RESERVED. HOWEVER, THE EXATRON STRINGY FLOPPY ROMS RESIDE HERE AND THERE ARE CURRENTLY A FEW ROM ADD ON DEVICES WHICH USE THIS AREA OF MEMORY.**
37DEH - 37ECH	**MEMORY MAPPED INPUT/OUTPUT LOCATIONS**
37DEH	DOS communication status address.
37DFH	DOS communication data address.
37E0H	Interrupt latch address.
37E1H	Disk drive select latch address.
37E4H	Cassette drive select latch address.
37E8H	Line printer port address.
37ECH	Floppy disk controller address.
3800H - 3BFFH	**KEYBOARD MEMORY**

The following bits will be set for each key that is pressed:

	0	1	2	3	4	5	6	7
3801H	@	A	B	C	D	E	F	G
3802H	H	I	J	K	L	M	N	O
3804H	P	Q	R	S	T	U	V	W
3808H	X	Y	Z					
3810H	0	1	2	3	4	5	6	7
3820H	8	9	:	;	,	-	.	/
3840H	CR	CL	BR	UA	DA	LA	RA	SP
3880H	SH							

Where

CR = CARRIAGE RETURN	UA = UP ARROW	RA = RIGHT ARROW
CL = CLEAR	DA = DOWN ARROW	SP = SPACE
BR = BREAK	LA = LEFT ARROW	SH = SHIFT

3C00H - 3FFFH	**VIDEO MEMORY**
4000H - 4014H	**RST VECTORS**
4000H - 4002H	RST 0008H
4003H - 4005H	RST 0010H
4006H - 4008H	RST 0018H
4009H - 400BH	RST 0020H
400CH - 400EH	RST 0028H
400FH - 4011H	RST 0030H
4012H - 4014H	RST 0038H
4015H - 401CH	**KEYBOARD DEVICE CONTROL BLOCK**
4015H	Device type.
4016H - 4017H	Driver address.
401BH - 401CH	RAM buffer address.
401DH - 4024H	**VIDEO DEVICE CONTROL BLOCK**
401DH	Device type.
401EH - 401FH	Driver address.
4020H - 4021H	Cursor location.
4022H	Cursor on/off flag.
4023H - 4024H	RAM buffer address.
4025H - 402CH	**PRINTER DEVICE CONTROL BLOCK**
4025H	Device type.
4026H - 4027H	Driver address.
4028H	Lines per page.
4029H	Lines printed so far.
402BH - 402CH	RAM buffer address.
402DH - 4035H	**LEVEL II SYSTEM VARIABLES**
402DH - 4035H	Used by DOS.
4036H - 403CH	Keyboard work area.
403DH - 4040H	Used by DOS.

4041H	Seconds - Real Time Clock.
4042H	Minutes - Real Time Clock.
4043H	Hours - Real Time Clock.
4044H	Year - Real Time Clock.
4045H	Day - Real Time Clock.
4046H	Month - Real Time Clock.
4047H - 407FH	Used by DOS.
4080H - 408DH	Division support routine.
408EH - 408FH	Starting address for USR call.
4093H - 4095H	INP routine.
4096H - 4098H	OUT routine.
4099H	Last key pressed.
409AH	RESUME flag.
409BH	Printer carriage position.
409CH	Current output device flag: −1 - cassette, 0 - video, & 1 - printer.
409DH	Size of line on the video display.
409EH	Size of line on the printer.
40A0H - 40A1H	Start of string space pointer.
40A2H - 40A3H	Current BASIC line number.
40A4H - 40A5H	Start of BASIC program pointer.
40A6H	Current cursor line position.
40A7H - 40A8H	Input Buffer pointer.
40A9H	Cassette input flag.
40AAH - 40ADH	Random number seed.
40AEH	LOCATE/CREATE variable flag.
40AFH	Current number type flag.
40B0H	Temporary storage location.
40B1H - 40B2H	MEMORY SIZE? pointer.

40B3H - 40B4H	Next available location in the temporary string work area pointer.
40B5H - 40D2H	Temporary string work area.
40D3H - 40D5H	Used for temporary string VARPTR's.
40D6H - 40D7H	Next available location in string space pointer.
40D8H - 40D9H	Temporary storage location.
40DAH - 40DBH	DATA line number.
40DCH	FOR flag.
40DDH	INPUT flag.
40DEH	READ flag.
40DFH - 40E0H	Used by DOS.
40E1H	AUTO flag.
40E2H - 40E3H	Current BASIC line number.
40E4H - 40E5H	AUTO increment.
40E6H - 40E7H	Temporary storage location.
40E8H - 40E9H	Stack pointer pointer.
40EAH - 40EBH	Line number with error.
40ECH - 40EDH	EDIT line number.
40EEH - 40EFH	Used by RESUME.
40F0H - 40F1H	Used by ONERROR.
40F2H	Error flag.
40F3H - 40F4H	Temporary storage location.
40F5H - 40F6H	Last line number executed.
40F7H - 40F8H	Last byte executed.
40F9H - 40FAH	Simple variables pointer.
40FBH - 40FCH	Array variables pointer.
40FDH - 40FEH	Free memory pointer.
40FFH - 4100H	READ pointer.
4101H - 411AH	Variable Declaration Table.

411BH		TRON/TROFF flag.
411CH		Used by floating point routines.
411DH - 4124H		REG1.
4125H - 4126H		Used by floating point routines.
4127H - 412EH		REG2.
4130H - 4149H		Internal print buffer.
414AH - 4151H		Used by floating pointer routines.
4152H - 41A3H		Disk Basic links.
41A6H - 41E4H		DOS links.
41E8H - 42E7H		Input Buffer area.
42E8H		Always zero.
42E9H		Start of user RAM.

Index

Index

T

U

V

W

X

Z

ISBN 0-8306-0275-5